Analyzing Geospatial Forest Data with GeoPandas and Rasterio

By Laya Zeinali

Laya Zeinali Yadegari
Ph.D. candidate at Tarbiat Modares University

About my research…

Forest ecosystem focus

Ph.D. research on forest ecosystem services and carbon cycle

Remote sensing

Using satellite data to model forest ecosystem and biomass estimation

Technical approach

Applying Python, Gee and machine learning for geospatial modeling

Environmental impact

Supporting sustainable forest management and climate change mitigation

Why is geospatial analysis considered a vital tool in environmental research, and what makes its application particularly crucial for understanding, managing, and conserving forest ecosystems in the face of climate and land-use challenges?

Geospatial Forest Analysis

geospatial analysis enable visualization of forest structure, patterns, and changes that traditional methods miss. Essential for conservation and sustainable management

Monitoring and assessment

- Precise evaluation of forest ecosystem using GIS and remote sensing

Data Integration

- Combining satellite imagery, LIDAR, and field surveys for comprehensive analysis

Multi-scale Application

- Analyzing forest attributes from local and global scales.

Forest Biomass and Carbon

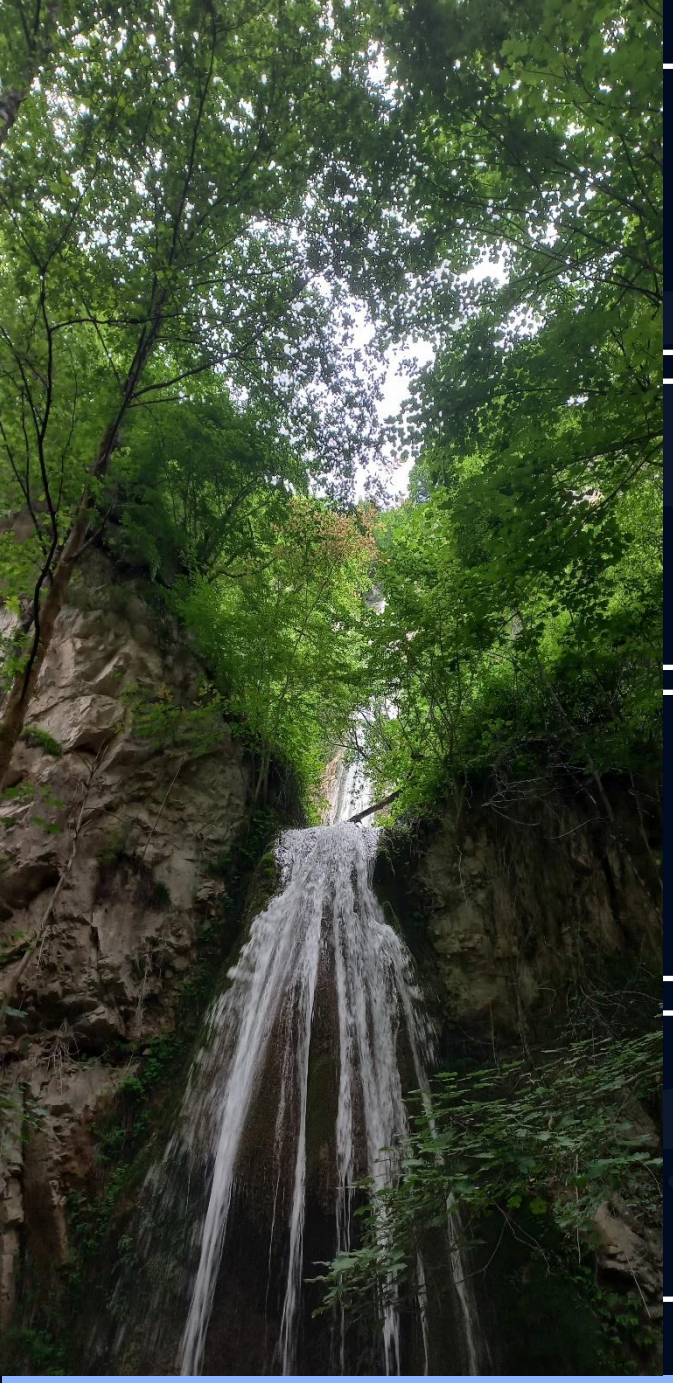
Timber production
Economic value of forest resources

Carbon sequestration
Forests store 76-98% of terrestrial organic carbon

Canopy height
Critical indicator for estimating biomass and carbon stocks

Climate regulation
Absorbs atmospheric greenhouse gases

- Forest ecosystems contain approximately 1146 gigatons of carbon across 17.4 billion hectares globally. One third of stored in vegetation, two-thirds in soil.



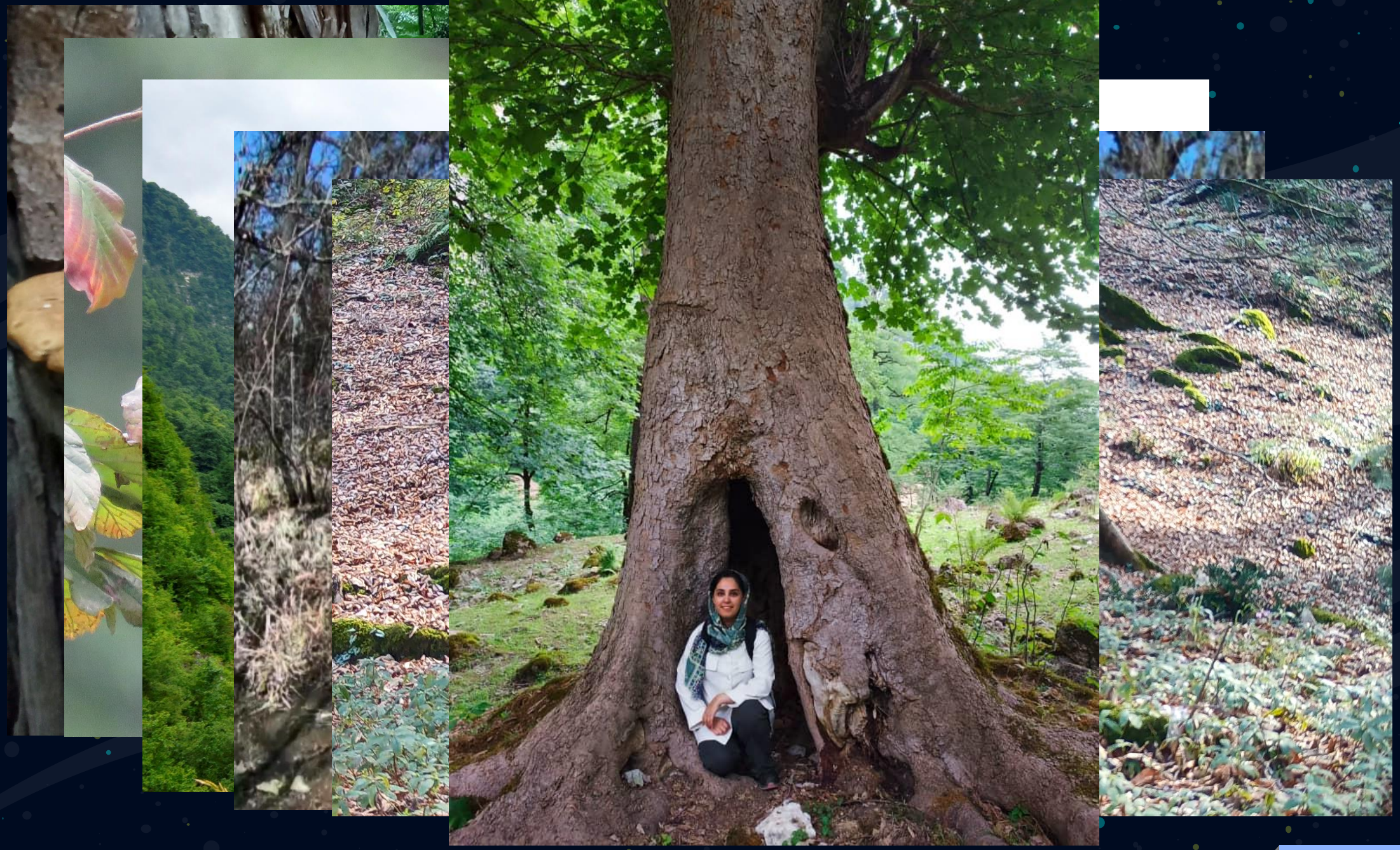
Ancient Ecosystem
Remnants of Arcto-Tertiary forests that survived the last glacial period. These forests stretch along the Caspian Sea in northern Iran.

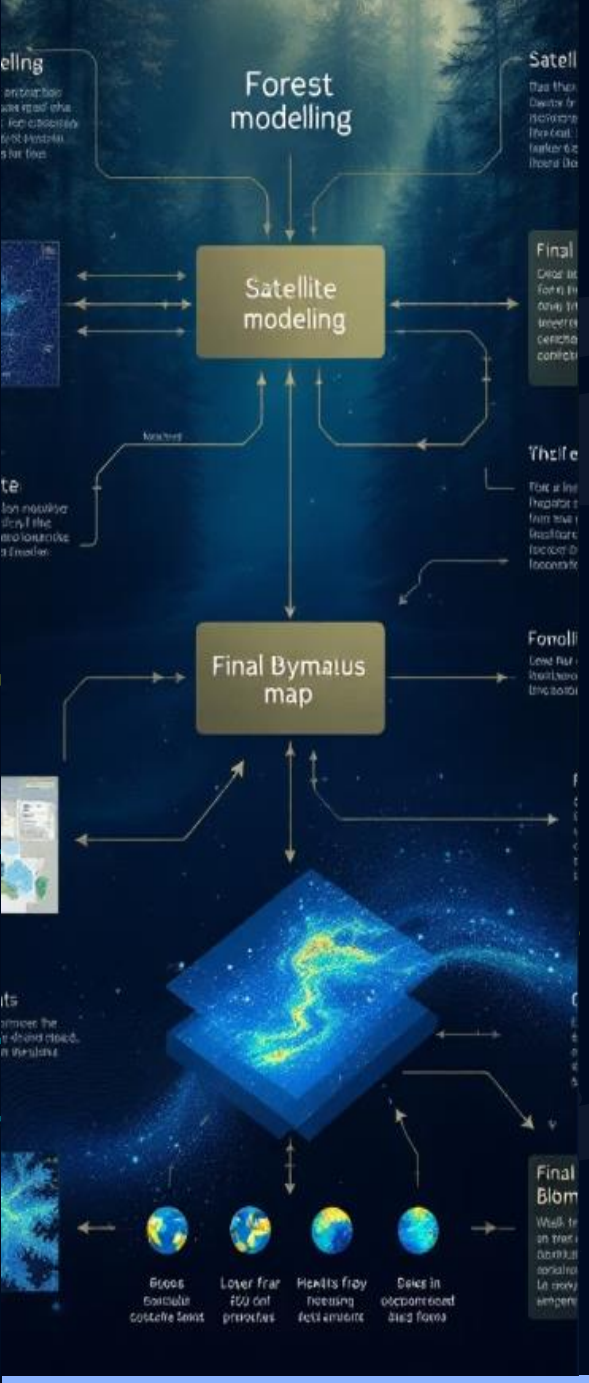
Rich Biodiversity
Home to over 80 tree species, including Oriental beech, hornbeam, and Persian ironwood. The forests also shelter endangered species such as the Persian leopard.

Climate Characteristics
Humid and temperate climate with annual precipitation ranging from 530 mm to 2,000 mm. There is no dry season in the western parts.

Conservation Status
Recognized as a UNESCO World Heritage Site, the Hyrcanian Forests face threats from deforestation, overgrazing, and urban expansion.

Hyrcanian forest

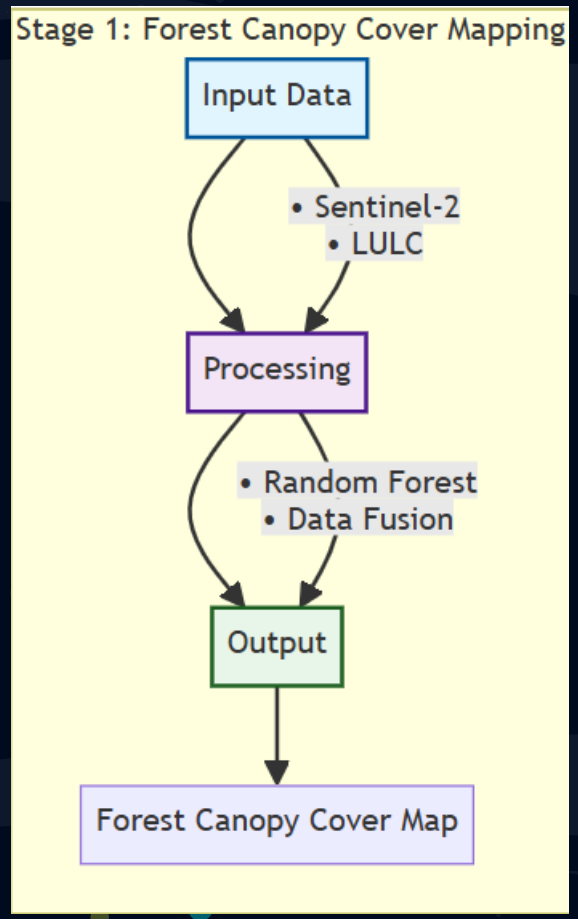


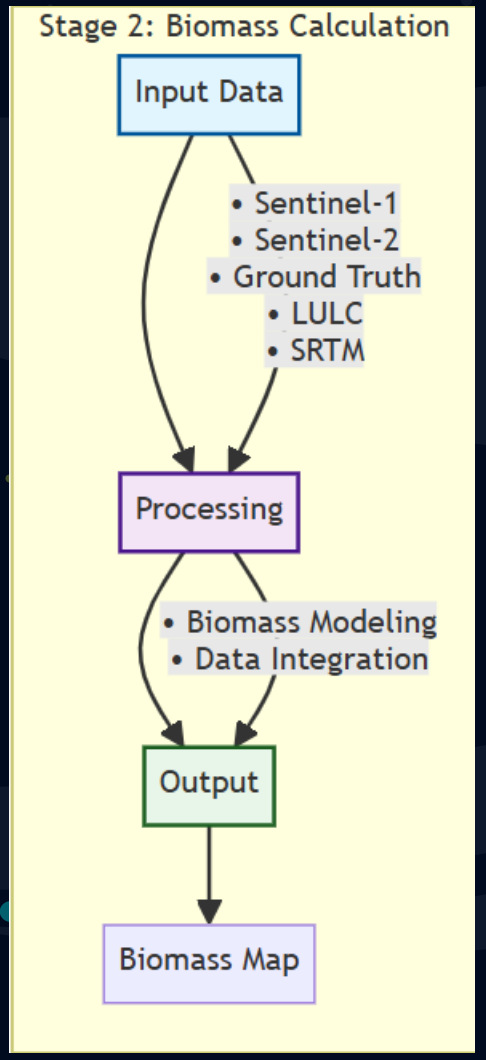


Data Sources Used

Icon	Data Source	Key Features	Application
	Sentinel-1 ✔ Used	SAR radar; all-weather, day/night imaging	Flood mapping, ground deformation (InSAR)
	Sentinel-2 ✔ Used	10–60 m spatial resolution; 13 spectral bands	Vegetation monitoring, land cover change
	SRTM ✔ Used	30 m global Digital Elevation Model	Slope analysis, drainage/basin extraction
	LULC ✔ Used	Land use/land cover classification	Urbanization assessment, habitat loss
	Ground Truth ✔ Used	Field surveys, GPS points	Validation and calibration
	GEDI (NASA ISS LiDAR) ✔ Used	Full-waveform LiDAR; RH metrics, vertical structure; ~25 m footprint	Canopy height/biomass estimation, carbon mapping, habitat assessment

An Integrated Approach to Modeling Forest Canopy Height and Aboveground Biomass Using GEDI, Sentinel-1, Sentinel-2, SRTM, and LULC Data





Why Python for Geospatial Analysis?

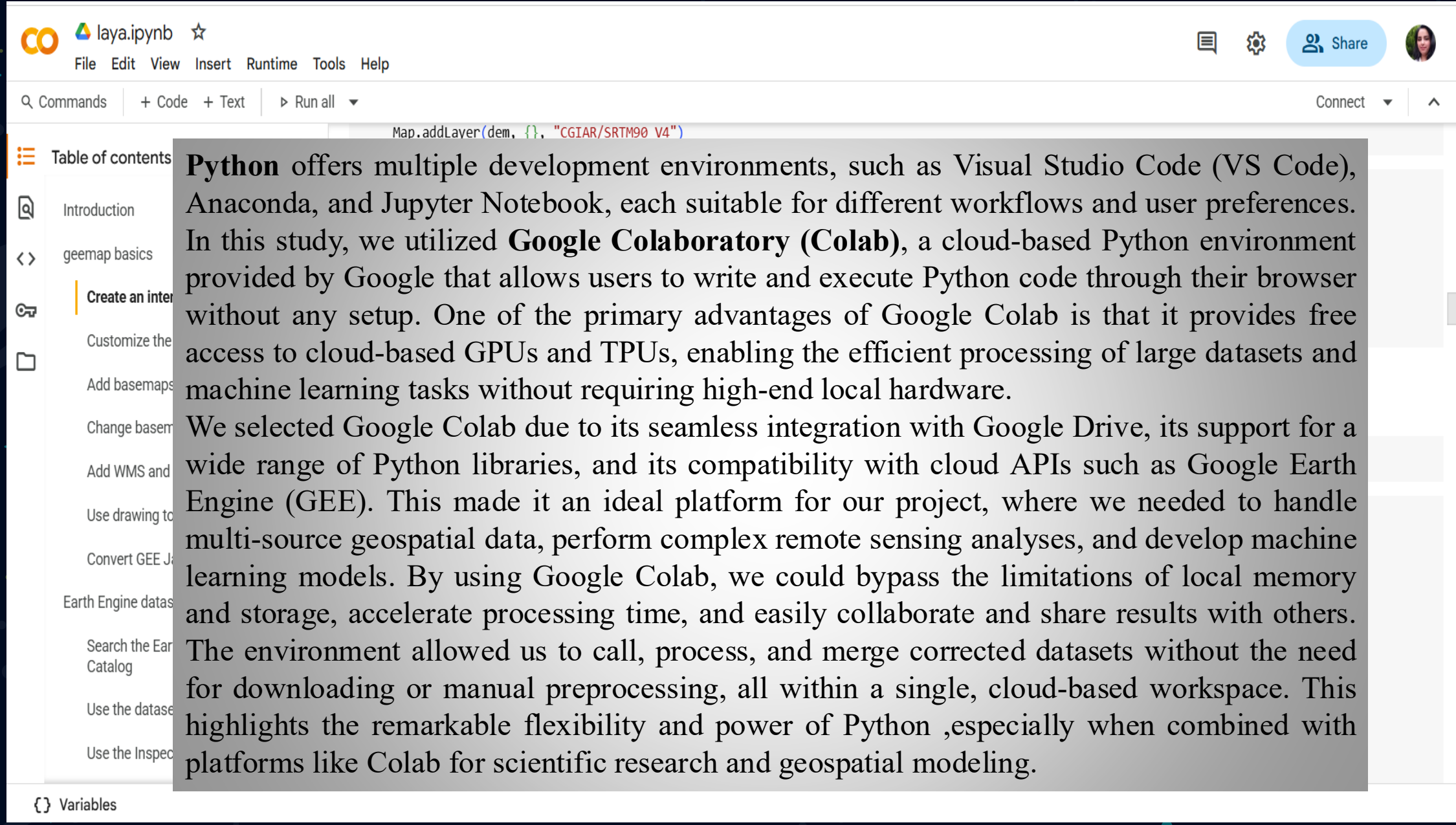
Powerful Integration
Seamless connection between platforms and data sources.

Strong Community
Extensive support, tutorials, and code examples.

Open-Source Ecosystem
Free libraries like GDAL, GeoPandas, and Rasterio.

GIS Integration
Works with QGIS, ArcGIS, and Google Earth Engine.

Python's simplicity and accessibility make it ideal for geospatial analysis. Its interpreted nature provides clear error messages, making it beginner-friendly compared to other languages.



Python offers multiple development environments, such as Visual Studio Code (VS Code), Anaconda, and Jupyter Notebook, each suitable for different workflows and user preferences. In this study, we utilized **Google Colaboratory (Colab)**, a cloud-based Python environment provided by Google that allows users to write and execute Python code through their browser without any setup. One of the primary advantages of Google Colab is that it provides free access to cloud-based GPUs and TPUs, enabling the efficient processing of large datasets and machine learning tasks without requiring high-end local hardware.

We selected Google Colab due to its seamless integration with Google Drive, its support for a wide range of Python libraries, and its compatibility with cloud APIs such as Google Earth Engine (GEE). This made it an ideal platform for our project, where we needed to handle multi-source geospatial data, perform complex remote sensing analyses, and develop machine learning models. By using Google Colab, we could bypass the limitations of local memory and storage, accelerate processing time, and easily collaborate and share results with others. The environment allowed us to call, process, and merge corrected datasets without the need for downloading or manual preprocessing, all within a single, cloud-based workspace. This highlights the remarkable flexibility and power of Python ,especially when combined with platforms like Colab for scientific research and geospatial modeling.

CHM7.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

Install Libraries

Install all the necessary libraries.

```
# Install some packages
!pip install rasterio # designed for reading and manipulating ras
!pip install earthpy # is a helper library built to simplify comm
```

Show hidden output

```
# Import the libraries
import geopandas as gpd
import numpy as np
import rasterio
from rasterio.features import rasterize
import pandas as pd
import earthpy.plot as ep
import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import make_regression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import cross_val_score
from google.colab import drive
```

The first step involves loading the essential libraries required for both phases, which includes the following:

pandas

GeoPandas

NumPy

seaborn

geemap

EarthPy

Rasterio

Matplotlib

scikit learn

Fiona

cartopy

Setting-up Colab

Mount your Google Drive

First, mount your Google Drive using the code below. Import the raster and vector datasets and prepare the features and labels.

```
[ ] # Mount Google Drive
    drive.mount('/content/drive')
```

Show hidden output

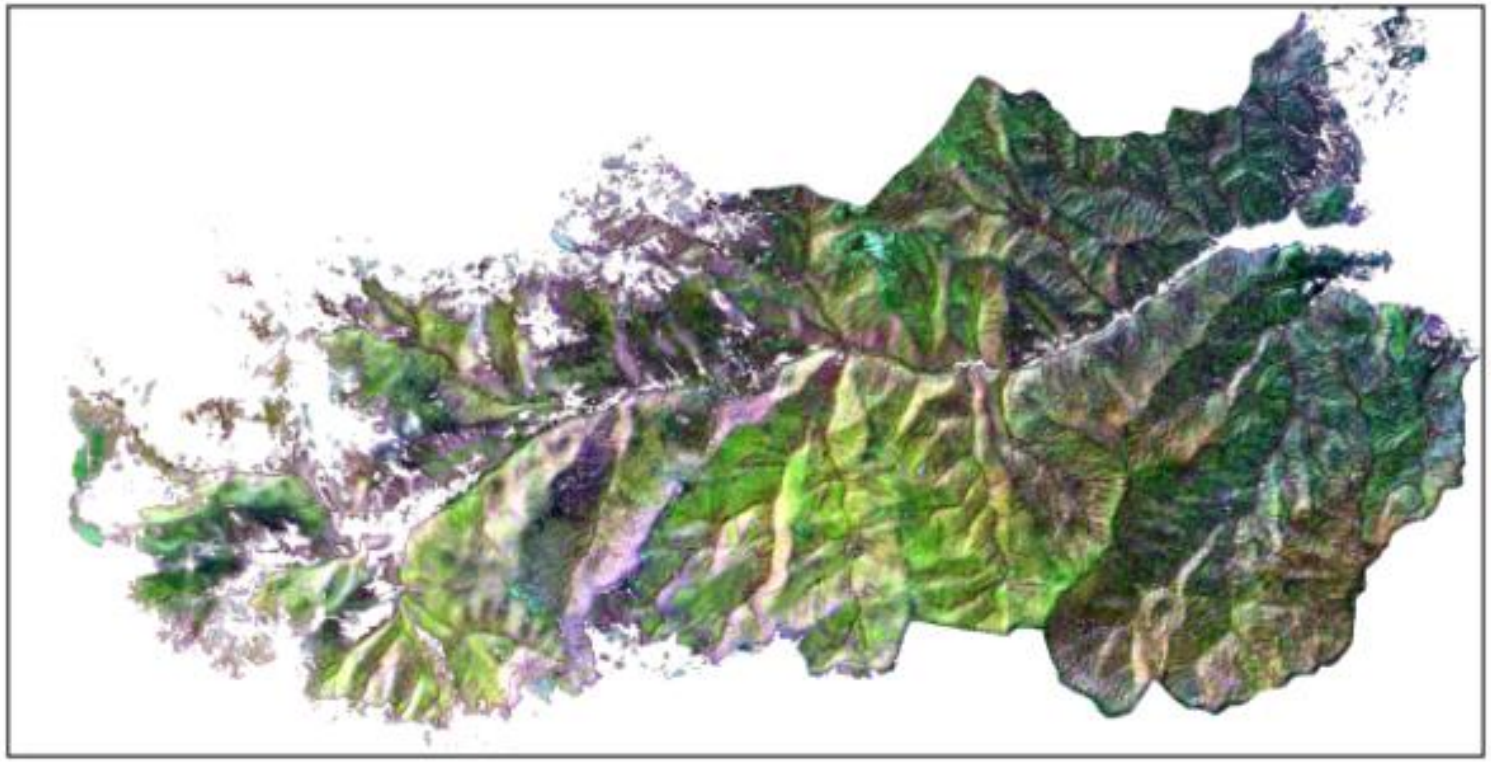
Define variables

Set the access path to both the vector and image datasets. Then define the response and predictor variables.

```
[ ] # Define path to datasets
    SAMPLE_PATH = '/content/drive/MyDrive/CHMinputs/chm-inputs73g.csv'
    IMAGE_PATH = '/content/drive/MyDrive/CHMinputs/chm-inputs73g.tif'
    SHP_PATH = '/content/drive/MyDrive/CHMinputs/7lulc-shp.shp'

    # Define varaibales
    Predictors = ['B2','B3', 'B4', 'B5', 'B6', 'B7', 'B8', 'B11', 'B12','elevation', 'slope', 'LULC', 'NDVI','SAVI','EVI','CCCI','MCARI2'] # Sentinel-2, SRTM
    Response = ['rh95','elev_highestreturn','elev_lowestmode'] # GEDI forest conopy height
```

In this section, since we are importing the data from Google Drive, we first need to connect Colab to Google Drive.
The data we import includes the boundary of the study area, which is provided in shapefile (SHP) format, Sentinel-2 data along with vegetation indices, and spatial LiDAR data that has already been combined. These datasets are available in two formats: Excel tables and raster images (tif).

```
Number of bands: 31
Image dimensions (Height x Width): 1682 x 3312
Coordinate Reference System: EPSG:32639
/usr/local/lib/python3.11/dist-packages/earthpy/spatial.py:561: RuntimeWarning: invalid value encountered in cast
  return (bytedata.clip(low, high) + 0.5).astype("uint8")
```



```
<Axes: >
```

7th watershed of 92, Hycanian forest

Prepare datasets for modeling

Import the sample data

Next, we are going to load the sample data with the canopy height (CH) and raster variables.

```
[ ] # Read sample
    samples = pd.read_csv(SAMPLE_PATH)[Predictors + Response]
    samples
```

	B2	B3	B4	B5	B6	B7	B8	B11	B12	elevation	slope	LULC	NDVI	SAVI	EVI	CCCI	MCARI2	rh95	elev_highestreturn	elev_lowestmode
0	0.1296	0.1553	0.1317	0.1899	0.3816	0.4606	0.4848	0.2911	0.1869	13	7	2	0.572749	0.474384	0.677475	0.437083	0.066530	9.73	29.575043	16.992796
1	0.1285	0.1502	0.1303	0.1899	0.3816	0.4606	0.4272	0.2911	0.1869	13	7	2	0.532556	0.421135	0.596065	0.384541	0.068964	9.73	29.575043	16.992796
2	0.1306	0.1550	0.1325	0.1887	0.3858	0.4652	0.4972	0.2904	0.1873	13	6	2	0.579165	0.484244	0.694561	0.449774	0.063489	9.73	29.575043	16.992796
3	0.1325	0.1574	0.1314	0.1887	0.3858	0.4652	0.4836	0.2904	0.1873	13	6	2	0.572683	0.473812	0.688832	0.438643	0.066146	9.73	29.575043	16.992796
4	0.1284	0.1497	0.1304	0.1828	0.3506	0.4109	0.4060	0.2754	0.1841	11	4	2	0.513796	0.398881	0.562265	0.379076	0.058204	9.73	29.575043	16.992796
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
616	0.1335	0.1623	0.1402	0.2132	0.4032	0.4786	0.4876	0.3272	0.2222	45	18	7	0.553361	0.462050	0.654213	0.391553	0.086770	4.71	58.870743	51.193836
617	0.1482	0.1819	0.1611	0.2600	0.4079	0.4570	0.4972	0.3426	0.2534	39	2	7	0.510558	0.435250	0.621349	0.313259	0.123528	3.24	63.843803	58.582966
618	0.1482	0.1819	0.1611	0.2600	0.4079	0.4570	0.4972	0.3426	0.2534	39	2	7	0.510558	0.435250	0.621349	0.313259	0.123528	3.24	63.843803	58.582966
619	0.1426	0.1850	0.1548	0.2497	0.4043	0.4638	0.4956	0.3400	0.2385	39	2	7	0.523985	0.444367	0.628829	0.329934	0.118527	3.24	63.843803	58.582966
620	0.1331	0.1616	0.1422	0.2178	0.4042	0.4573	0.4692	0.3116	0.2169	38	3	7	0.534838	0.441335	0.617377	0.365939	0.089846	3.14	47.968143	41.486481

621 rows × 20 columns

As indicated by the column headers, this dataframe contains information from Sentinel-2, GEDI, land use/land cover data (LULC) used for masking forested areas, as well as vegetation indices.

The initial step in preparing the dataset for machine learning analysis involves partitioning it into training and testing subsets. In this study, an 80/20 split ratio was applied, allocating 80% of the data for training the model and 20% for testing its performance.

Splitting the dataset into training and testing subsets is an essential step in data preparation for machine learning. While the training set is used to fit the model, the testing set serves as the basis for evaluating the model's predictive performance.

Split training data

First, let's split the training points into training and test datase

```
[ ]  # Split into train and test
     train, test = train_test_split(samples, test_size=0.

     # Get varibles input and output
     X_train = train[Predictors].to_numpy()
     X_test = test[Predictors].to_numpy()
     y_train = train[Response].to_numpy().astype(float)
     y_test = test[Response].to_numpy().astype(float)

     # Show the data shape
     print(f'Train features: {X_train.shape}\nTest featu
```

```
Train features: (496, 17)
Test features: (125, 17)
Train label: (496, 3)
Test label: (125, 3)
```

Check for missing or NAs values.

```
[ ]  # Check for NaN or missing values in X_train, X_test, y_train, y_test
     def check_for_missing_values(array, array_name):
         nan_count = np.isnan(array).sum()
         if nan_count > 0:
             print(f"{array_name} contains {nan_count} missing values (NaNs).")
         else:
             print(f"{array_name} does not contain any missing values.")

     # Check X_train, X_test, y_train, y_test for missing values
     check_for_missing_values(X_train, 'X_train')
     check_for_missing_values(X_test, 'X_test')
     check_for_missing_values(y_train, 'y_train')
     check_for_missing_values(y_test, 'y_test')
```

```
X_train does not contain any missing values.
X_test does not contain any missing values.
y_train does not contain any missing values.
y_test does not contain any missing values.
```

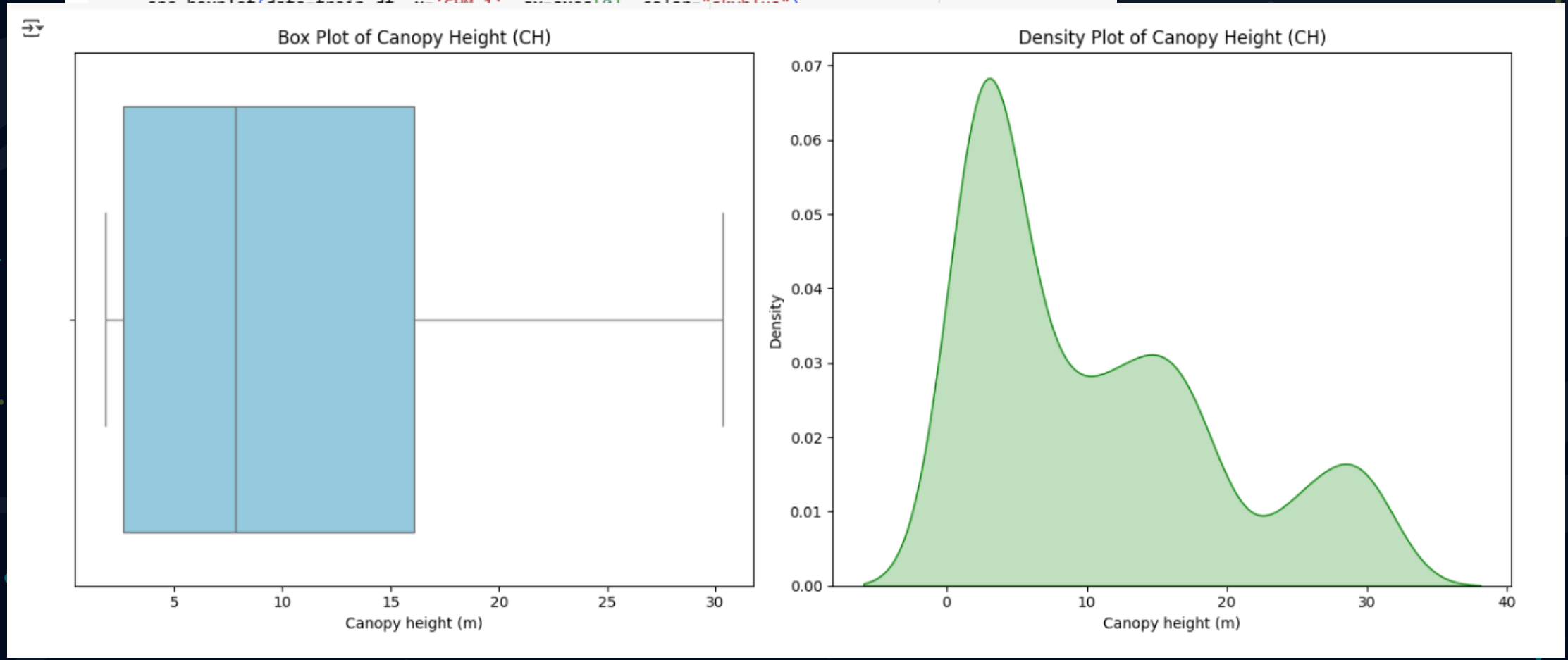
Perform Exploratory data analysis (EDA) is an important step in understanding your data before building a machine learning model. To perform EDA on your training dataset, we will use libraries such as pandas for data manipulation and matplotlib or seaborn for data visualization. First, we will start by creating a DataFrame for the training dataset.

```
[ ]  import pandas as pd
     import numpy as np

     # Get the original co
     X_train_cols = Predic

     # Get the number of c
     num_y_cols = y_train.

     # Create column names
     y_train_cols = [f'CHM

     # Combine column name
     column_names = X_trai

     # Create the DataFram
     train_df = pd.DataFra

     # Display the DataFra
     print(train_df.head()
```

```
         B2      B3      B4      B5      B6      B7      B8     B11     B12  \
0  0.1321  0.1593  0.1370  0.1954  0.3460  0.4300  0.4416  0.2996  0.1997
1  0.1316  0.1495  0.1342  0.1916  0.3494  0.4121  0.4136  0.2903  0.1895
2  0.1280  0.1544  0.1285  0.1860  0.3991  0.4801  0.5076  0.2990  0.1897
3  0.1331  0.1590  0.1369  0.1976  0.3651  0.4245  0.4384  0.2737  0.1848
4  0.1459  0.1808  0.1609  0.2342  0.4281  0.4944  0.5196  0.3426  0.2329

   elevation  slope  LULC      NDVI      SAVI       EVI      CCCI    MCARI2  \
0       14.0    4.0   5.0  0.526443  0.423605  0.598264  0.386499  0.065238
1       56.0    5.0   2.0  0.510040  0.399981  0.567056  0.366821  0.064818
2       17.0    2.0   2.0  0.595975  0.500528  0.718755  0.463668  0.066439
3       23.0    6.0   2.0  0.524074  0.420580  0.597479  0.378616  0.068994
4       24.0    2.0   7.0  0.527112  0.455781  0.644796  0.378615  0.080988

       CHM_1      CHM_2      CHM_3
0  18.389999  39.621902  18.827667
1   6.610000  67.590225  59.030441
2  16.070000  43.250122  25.154976
3   1.940000  30.055077  27.395126
4   2.170000  44.163158  39.667469
```

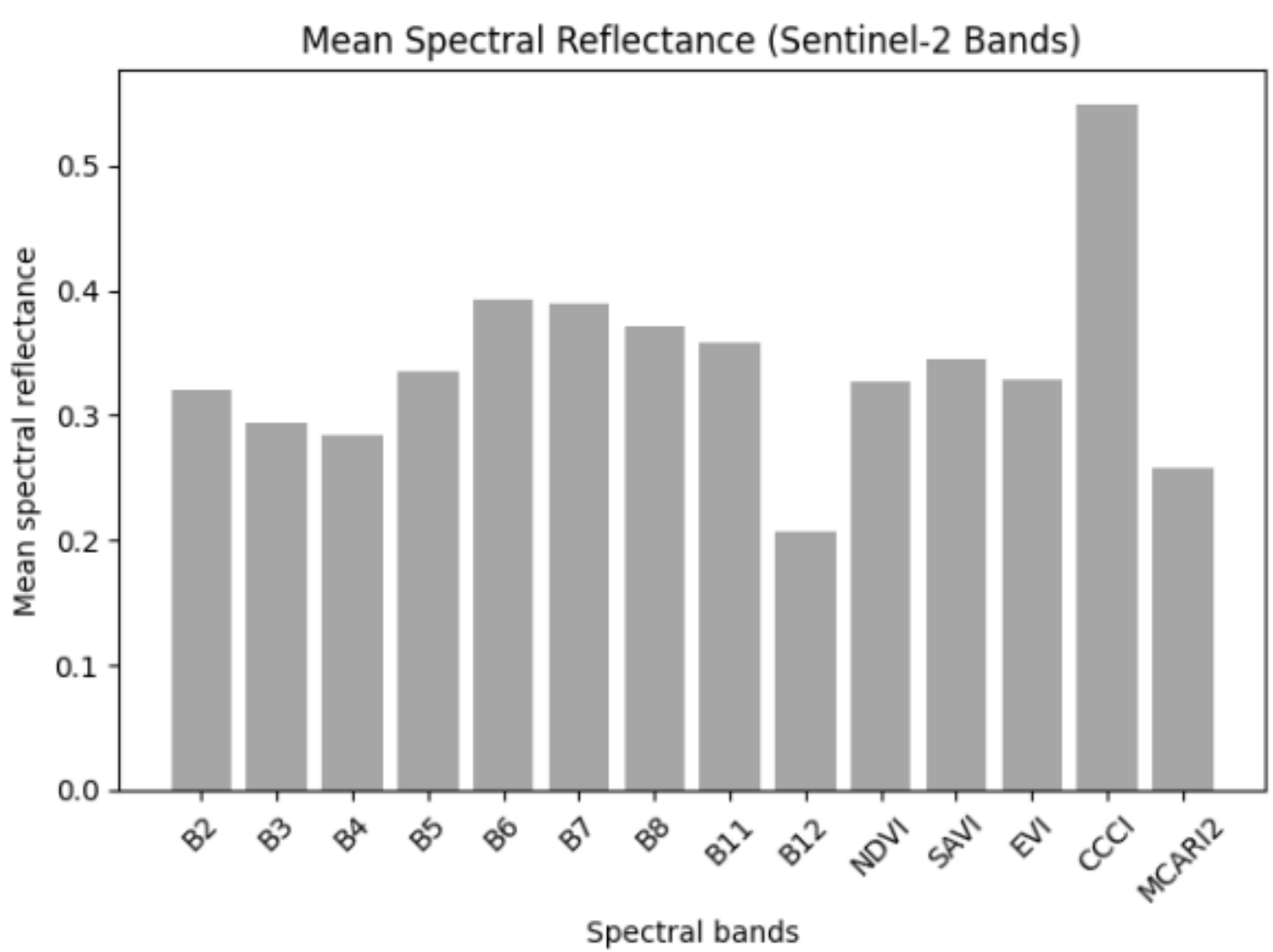
Next, create box and density plot for the canopy height.

```
[ ]  # Set up the figure and axes for box and density plots
     fig, axes = plt.subplots(1, 2, figsize=(14, 6))

     # Box plot for 'CHM' - Renamed column to 'rh95'
     # Use train_df['rh95'] as the input for the x argument, assuming 'rh95' is your target variable
```

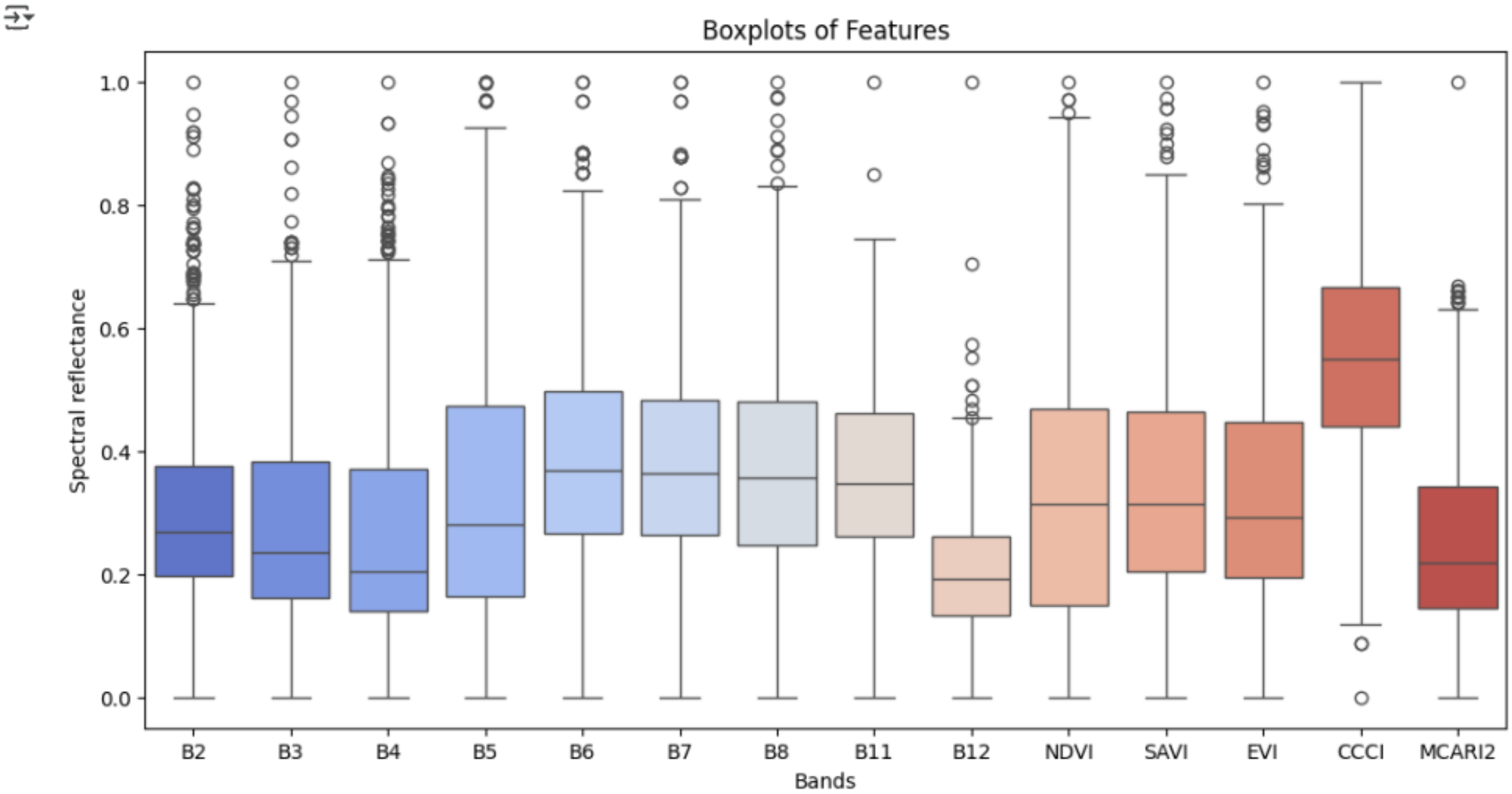


Let's generate a bar plot to show mean spectral reflectance for each band.

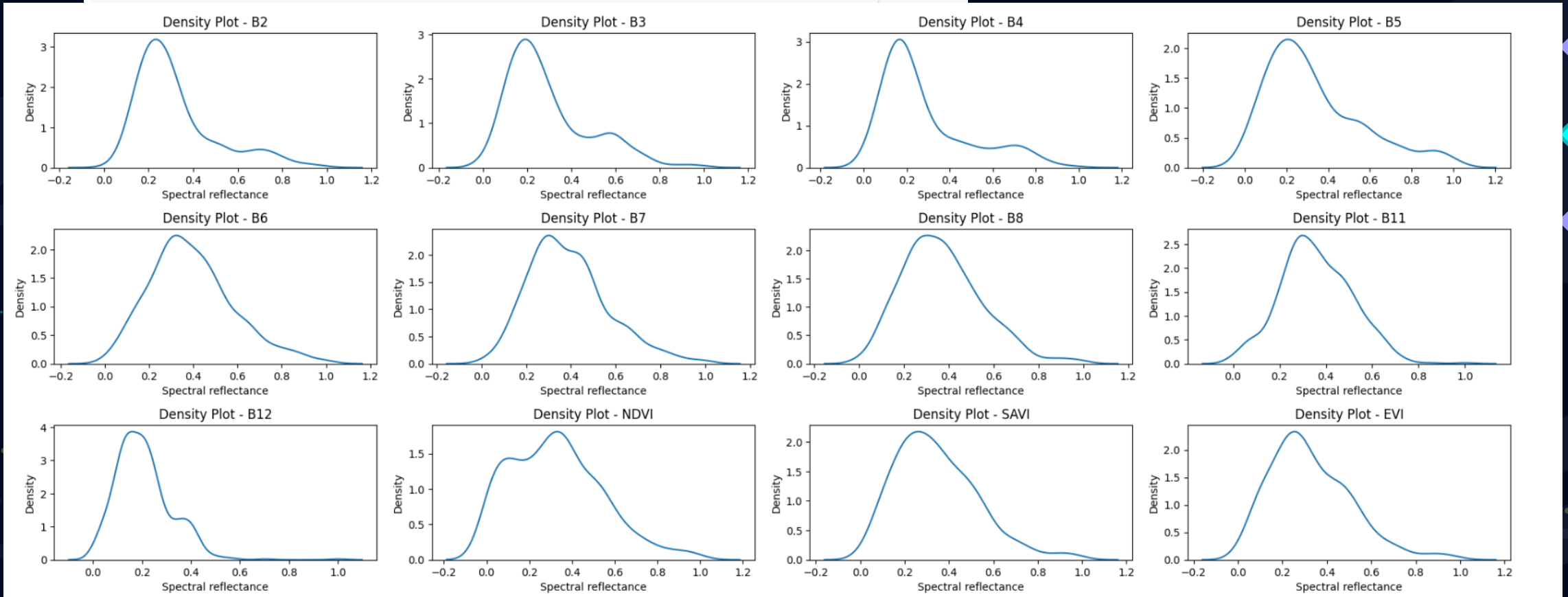
```
# Exclude non-spectral c
exclude_columns = ['elev
spectral_bands = [band f

# Normalize data
from sklearn.preprocessi
scaler = MinMaxScaler()
train_df[spectral_bands]

# Calculate mean reflect
mean_reflectance = [trai

# Plot the mean reflecta
plt.bar(spectral_bands,

# Add labels and title
plt.xlabel('Spectral ban
plt.ylabel('Mean spectra
plt.title('Mean Spectral

# Improve readability
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



We will visualize the distribution and potential outliers of each feature using boxplots. This code will generate a boxplot for each feature, showing the median, quartiles, and any potential outliers.



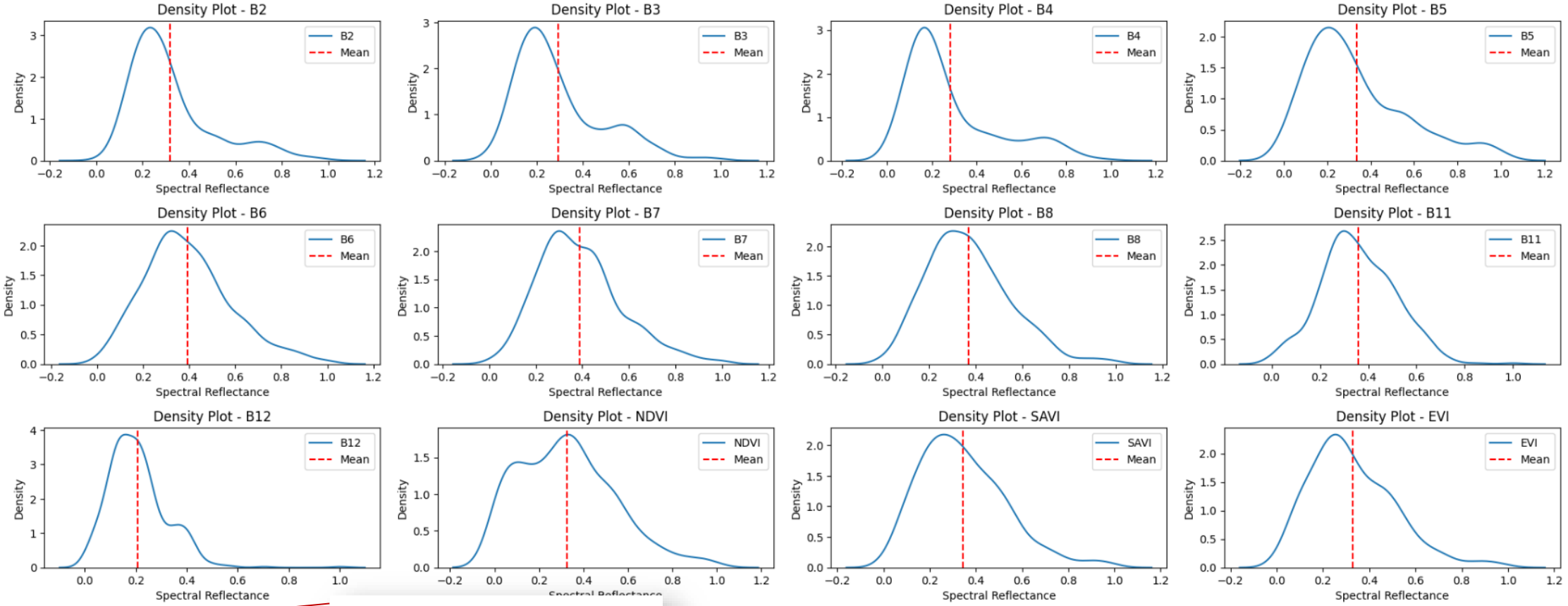
We will also visualize the distribution of each feature and the target variable using density plots (also known as kernel density plots).




```
plt.tight_layout()
plt.show()
```

```
[ ]
    # Drop unnecessary columns for analysis
```

```
Features with low variance (consider removing): []
```




```
Feature 'B2' has 47 outliers.
Feature 'B3' has 16 outliers.
Feature 'B4' has 32 outliers.
Feature 'B5' has 8 outliers.
Feature 'B6' has 14 outliers.
Feature 'B7' has 11 outliers.
Feature 'B8' has 9 outliers.
Feature 'B11' has 2 outliers.
Feature 'B12' has 9 outliers.
Feature 'NDVI' has 4 outliers.
Feature 'SAVI' has 9 outliers.
Feature 'EVI' has 10 outliers.
Feature 'CCCI' has 3 outliers.
Feature 'MCARI2' has 11 outliers.
```


```
Feature 'B2' has 47 outliers.
Feature 'B3' has 16 outliers.
Feature 'B4' has 32 outliers.
Feature 'B5' has 8 outliers.
Feature 'B6' has 14 outliers.
Feature 'B7' has 11 outliers.
Feature 'B8' has 9 outliers.
Feature 'B11' has 2 outliers.
Feature 'B12' has 9 outliers.
Feature 'NDVI' has 4 outliers.
Feature 'SAVI' has 9 outliers.
Feature 'EVI' has 10 outliers.
Feature 'CCCI' has 3 outliers.
Feature 'MCARI2' has 11 outliers.
```

Select and Train a Model Training
Train random forest (RF) model
We will use the random forest, which was introduced in Scikit-learn 0.21.

```
[ ]  # Set a random seed for reproducibility
     np.random.seed(42)

     # Create and fit the Random Forest model
     rf_reg = RandomForestRegressor()
     # Reshape y_train to have shape (n_samples, n_targets) instead of flattening it
```

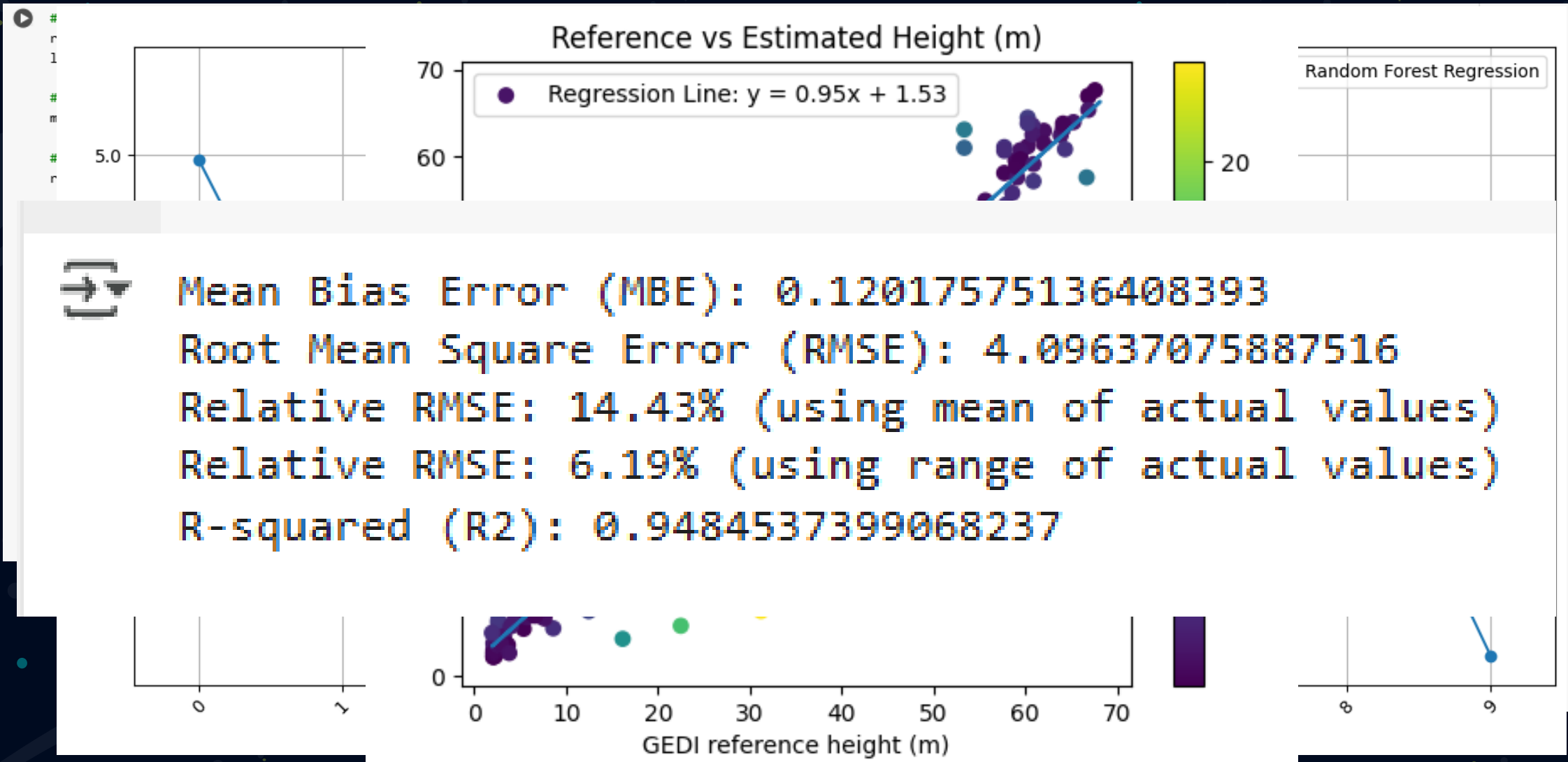
Finally, let's check the rmse score for the RF regression model.

```
[ ]  # Check the RMSE scores
     rf_reg_rmse_scores = np.sqrt(-scores)
     print(rf_reg_rmse_scores)
```

```
[4.98063995 3.93136729 4.45676327 3.67958969 5.29239349 4.93356844
 3.7980488  4.23235086 4.26366591 3.13390939]
```

```
scoring= neg_mean_squared_error , cv=10)
```


Finally, let's plot a graph showing the performance of random forest regression model. Predict using the test data and create a scatter plot.




```
Mean Bias Error (MBE): 0.12017575136408393
Root Mean Square Error (RMSE): 4.096370758875l6
Relative RMSE: 14.43% (using mean of actual values)
Relative RMSE: 6.19% (using range of actual values)
R-squared (R2): 0.9484537399068237
```

In the final step, the canopy height map is produced, and the resulting output is exported and stored in Google Drive for further use and analysis.

Perform Explainable Machine Learning (xML) We will use SHAP (SHapley Additive exPlanations), a popular explainable machine learning (xML) method. SHAP is based on Shapley values, a concept from cooperative game theory introduced by Lloyd Shapley in 1953, which has been adapted for interpreting machine learning models. In XML, Shapley values provide a way to quantify each feature's contribution to a prediction by calculating its average marginal contribution across all possible subsets of features. In other words, Shapley values fairly distribute the "credit" for a prediction among different features, offering a comprehensive view of feature importance within machine learning.


```
[ ]  # Import the shap library.
     import shap
```

```
# Define vmin and vmax based on SHAP values
max_abs_shap = max([np.nanmax(np.abs(shap_maps[predictor])) for predictor in Predictors])
vmin = -max_abs_shap
vmax = max_abs_shap
```

```
Computing SHAP values in 558 batches
Computing SHAP values:   1%||          | 8/558 [14:32<16:39:51, 109.07s/it]
```

sights into understanding feature contributions.

ote that it takes times to run the SHAP using RF model. You can use 'shap.sample()' to create a summarized b
e computation of SHAP values.

```
# Create the SHAP explainer using the trained model
explainer_rf = shap.Explainer(rf_model.predict, X_train)

# Compute SHAP values for the training set
shap_values_rf = explainer_rf(X_train)
```

```
PermutationExplainer explainer:  17%|▉          | 71/421 [00:37<02:27,  2.37it/s]
```

layazeinali
22 November 2024 10:05

Waiting to finish the current execution.

After estimating the canopy height, the process proceeds to the calculation of forest biomass. For this purpose, a new notebook is created, the environment is initialized by importing the required libraries, and the analysis is carried out using the ground-based data.

Preparing the environment

```
!pip install matplotlib-scalebar
!pip install pandas geopandas rasterio shapely fic
!pip install cartopy
!pip install cartopy geopandas
!pip install contextily
```

Show hidden output

```
import pandas as pd
import geopandas as gpd
import rasterio
import shapely
import fiona
import matplotlib.pyplot as plt
import ee
import geemap
import cartopy
```

Preparing the field data

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import pandas as pd
# File paths
file_paths = [
    '/content/drive/My Drive/plot_tree.xlsx',
]

# Load files into a dictionary of DataFrames
dataframes = {}
for path in file_paths:
    # Extract file name for dictionary key
    file_name = path.split('/')[-1].replace('.xlsx', '')
    dataframes[file_name] = pd.read_excel(path)

# Accessing individual DataFrames
field_data = dataframes['plot_tree']
```

The expression field_data.columns.tolist() in Python is used to retrieve the column names of a pandas DataFrame (in this case, field_data) as a list.

```
field_data.columns.tolist()
```

```
['YEAR',
 'HFI',
 'PLOTID',
 'Basin',
 'X',
 'Y',
 'Zone',
 'Province',
 'Province_code',
 'OLD_ID',
 'landuse',
 'macro_radii',
 'Altitude',
 'Slope',
 'Aspect',
 'Soil_depth',
 'Litter_thickness',
 'Drainage',
 'Erosion',
 'Grazing',
 'number_strata',
 'Stand_origin',
 'Forest_type',
 'Regeneration_status',
 'canopy_cover',
```

field_data.columns returns an Index object containing the column names of the DataFrame.

tolist() converts that Index into a standard Python list.

['YEAR', 'HFI', 'PLOTID', 'Basin', 'X', 'Y', 'Zone', 'Province', 'Province_code', 'OLD_ID', 'landuse', 'macro_radii', 'Altitude', 'Slope', 'Aspect', 'Soil_depth', 'Litter_thickness', 'Drainage', 'Erosion', 'Grazing', 'number_strata', 'Stand_origin', 'Forest_type', 'Regeneration_status', 'canopy_cover', 'kooreh_zoghal', 'Bedrock', 'Cause_1', 'Cause_2', 'Cause_3', 'Cause_5', 'Cause_6', 'Cause_4', 'Cause_7', 'Cause_8', 'Main_sp_first', 'Main_sp_second', 'Companion_sp', 'area', 'Landuse_name', 'Landuse_forest', 'Reclaim', 'Azimuth', 'Landslide', 'Cause_nogeneration', 'Meso_radii', 'Micro_radii', 'Nano_radii', 'litter_cover', 'Litter_decomposition', 'Succession', 'Moss_cover', 'Regeneration_origion', 'Smagling', 'naturalness', 'winthrow', 'fire', 'fire_type', 'fire_severity', 'cause_of_damages', 'disease', 'upload_date', 'Control', 'control_type', 'Authority', 'Team', 'code_in_database', 'Area', 'landuse_name', 'forest_nonforest', 'inventory_status', 'canopy_cover_recoded', 'Soil_depth_label', 'Drainage_label', 'Bedrock_label', 'Litter_thickness_label', 'Litter_cover_label', 'Litter_decomposition_label', 'Main_sp_first_30', 'Main_sp_second_30', 'Companion_sp_30', 'Main_sp_first_30_name', 'Main_sp_first_30_per_name', 'Main_sp_second_30_name', 'Main_sp_second_30_per_name', 'Companion_sp_30_name', 'Companion_sp_30_per_name', 'Stand_origin_label', 'Forest_type_label', 'number_strata_label', 'Succession_label', 'Moss_cover_label', 'Regeneration_status_label', 'Cause_9', 'Cause_10', 'Erosion_label', 'Grazing_label', 'Landslide_label', 'Disease_1', 'Disease_2', 'Disease_3', 'Disease_4', 'Disease_5', 'Disease_6', 'v_fallen_tree', 'v_illegal_logging', 'v_legal_logging', 'v_not_found', 'v_standing_tree', 'v_standing_tree_Binned', 'v_fallen_tree_Binned', 'd_stem_fallen_tree', 'd_stem_illegal_logging', 'd_stem_legal_logging', 'd_stem_not_found', 'd_stem_standing_tree', 'd_tree_fallen_tree', 'd_tree_illegal_logging', 'd_tree_legal_logging', 'd_tree_not_found', 'd_tree_standing_tree', 'biomass_fallen_tree', 'biomass_illegal_logging', 'biomass_legal_logging', 'biomass_not_found', 'biomass_standing_tree', 'v_growth_ha_year', 'max_h']

```
# Select relevar
data_for_SHNFI_H
    'PLOTID',
    'Zone',
    'X',
    'Y',
    'Slope',
    'Altitude',
    'canopy_cove
    'v_standing_
    'biomass_sta
    'max_h'
]].rename(columr
    'v_standing_
    'Slope': 'Sl
    'Altitude':
    'biomass_sta
})
data_for_SHNFI_H
```

```
import pandas as pd

# Replace NaN values in the 'max_h' and canopy_cover_recoded
data_for_SHNFI_HNFI6.update(
    data_for_SHNFI_HNFI6[["max_h", "canopy_cover_recoded"]].fillna({"max_h": 0, "canopy_cover_recoded": 6.0})
)
```

```
data_for_SHNFI_HNFI6.isna().sum()
```

	0
PLOTID	0
Zone	0
X	0
Y	0
Slope_field	0
Altitude_field	0
canopy_cover_recoded	0
Volume	0
Biomass	0
max_h	0

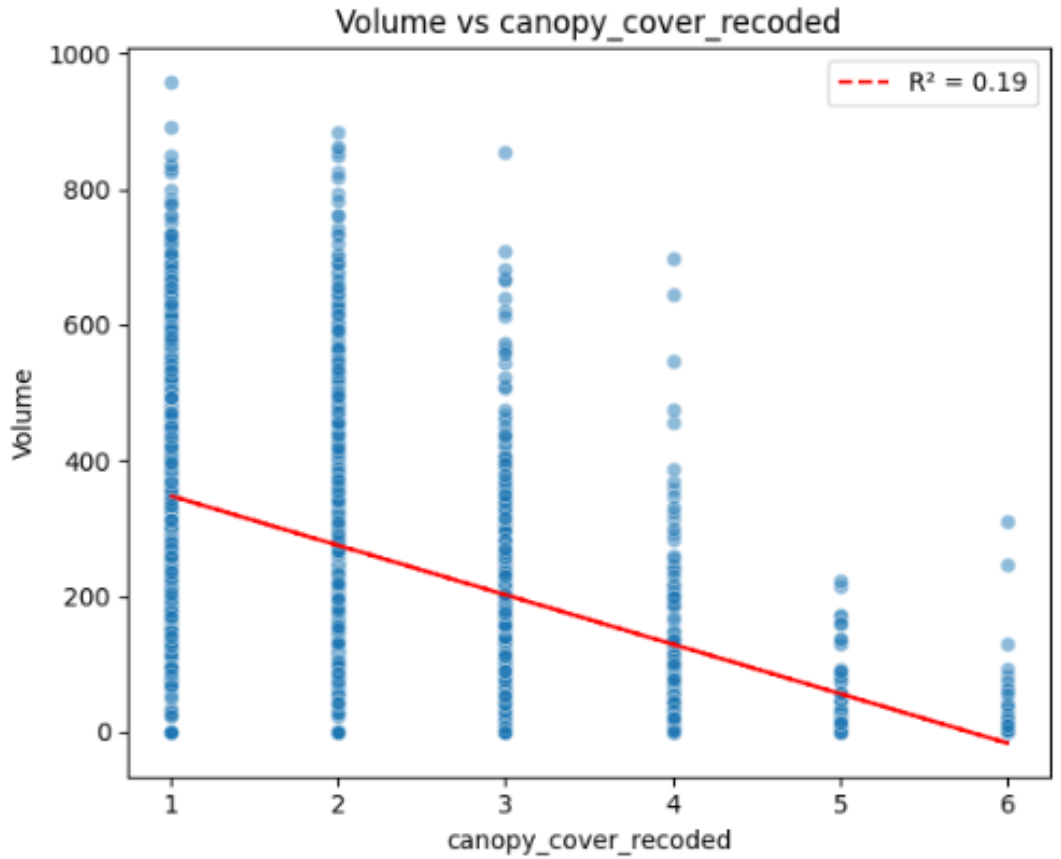
dtype: int64

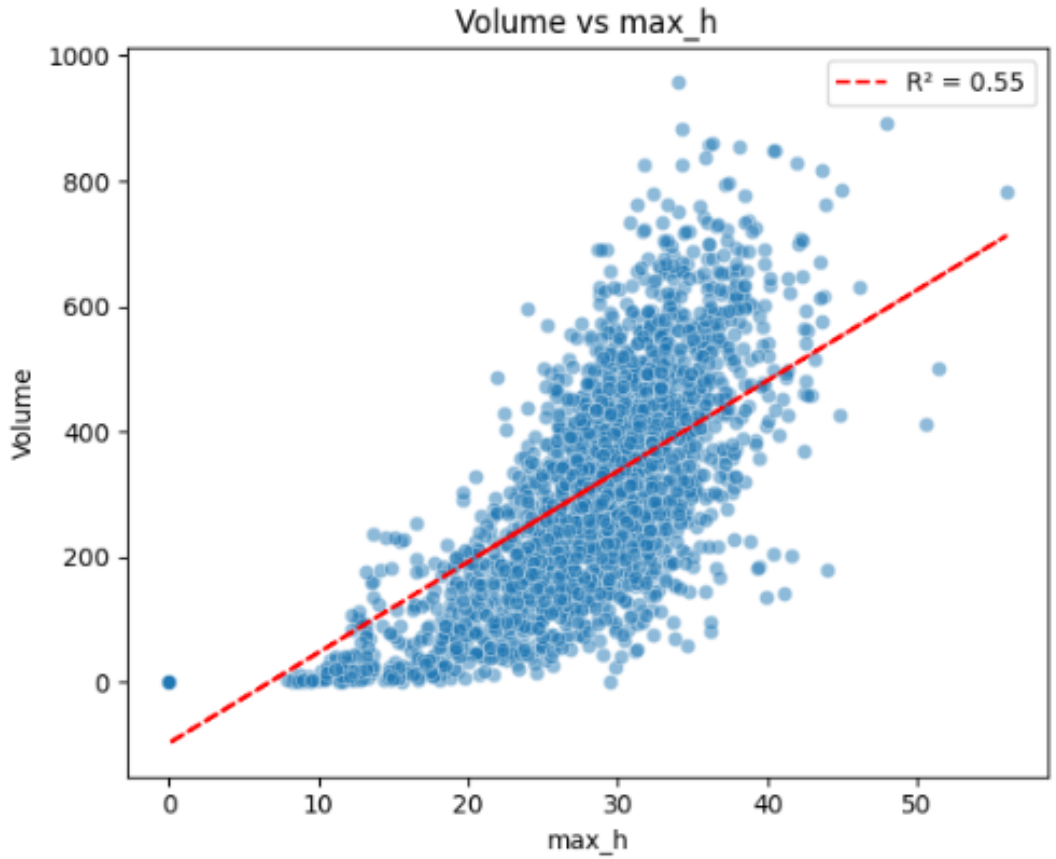
	PLOTID	Biomass	max_h
5847	A01002	172.277374	21.435173

✓ Our dataset includes 2,553 rows and 128 columns.

```
[ ]  import matplotlib.pyplot as plt
     import seaborn as sns
     from sklearn.linear_model import LinearRegression
     from sklearn.metrics import r2_score

     # Define x variables and y variable
     x_vars = ["canopy_cover_recoded", "max_h"]
     y_var = "Volume"
```

```
Using available columns: ['canopy_cover_recoded', 'max_h']
```





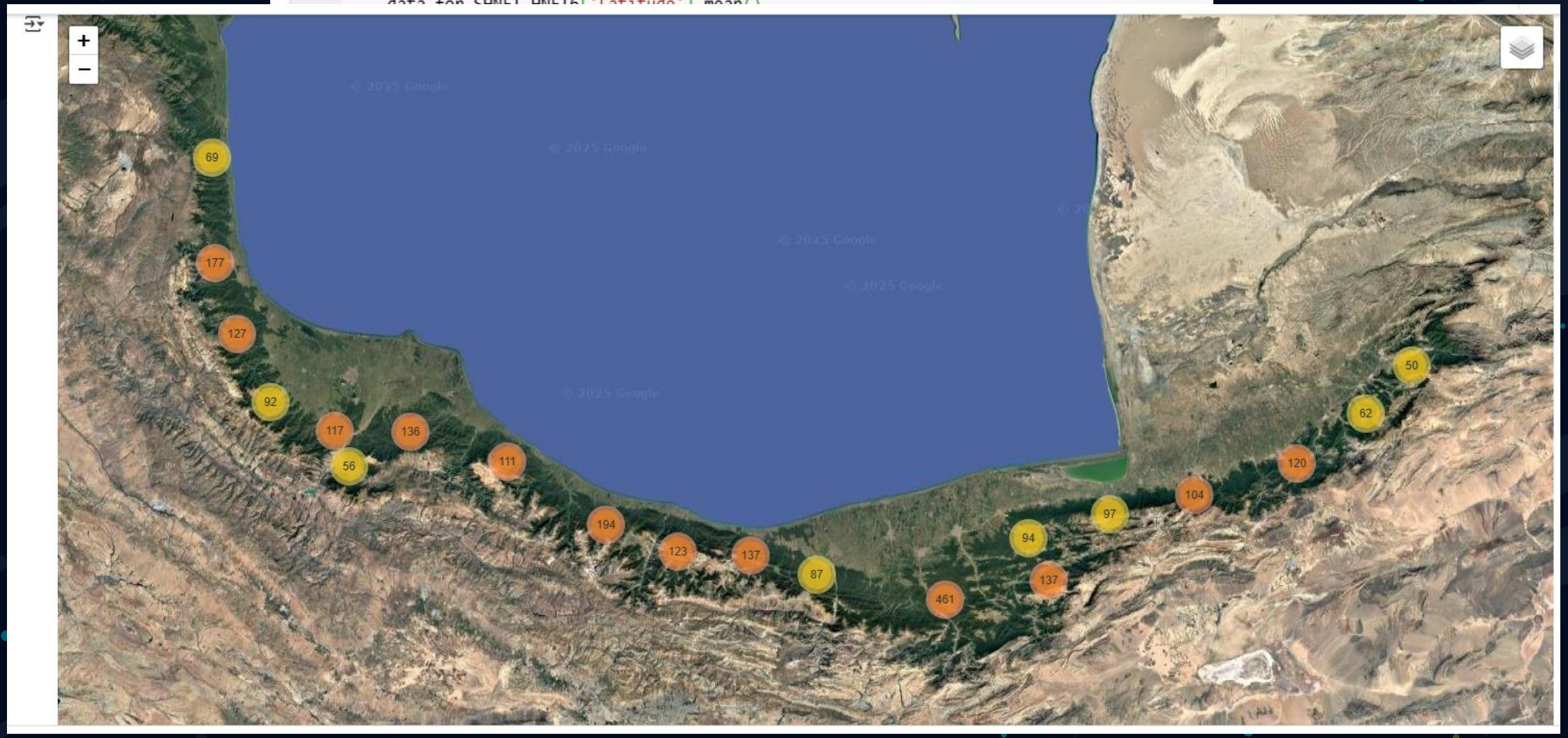
Code to Convert UTM Coordinates to EPSG:4326

```
[ ] import geopandas as gpd
    from shapely.geometry import Point

    # Step 1: Create Geometry Column from 'X' and 'Y' Coordinates
    data_for_SHNFI_HNFI6 = data_for_SHNFI_HNFI6.copy()  # Ensure you're working on a copy
    data_for_SHNFI_HNFI6['geometry'] = data_for_SHNFI_HNFI6.apply(
        lambda row: Point(row['X'], row['Y']), axis=1
    )

    # Step 2: Process Each Zone Separately and Transform to EPSG:4326
    gdfs = []
    for zone in [39, 40]:  # Handle UTM zones 39 and 40
        # Filter data for the current zone
        subset = data_for_SHNFI_HNFI6[data_for_SHNFI_HNFI6['Zone'] == zone].copy()  # Explic

        # Set CRS based on the zone
        crs = f"EPSG:326{zone}"  # UTM CRS for the given zone
        gdf = gpd.GeoDataFrame(subset, geometry='geometry', crs=crs)

        # Transform to WGS84 (EPSG:4326)
        gdf = gdf.to_crs("EPSG:4326")

        # Append the transformed GeoDataFrame
        gdfs.append(gdf)

    # Step 3: Concatenate All GeoDataFrames
    combined_gdf = gpd.GeoDataFrame(pd.concat(gdfs, ignore_index=True), crs="EPSG:4326")

    # Step 4: Extract Reprojected Coordinates
    data_for_SHNFI_HNFI6['Longitude'] = combined_gdf.geometry.x.values
    data_for_SHNFI_HNFI6['Latitude'] = combined_gdf.geometry.y.values

    data_for_SHNFI_HNFI6
```

This code takes point data with UTM coordinates (X, Y) from zones 39 and 40 and converts them into geographic coordinates (longitude and latitude) in the WGS84 system (EPSG:4326). First, it creates a geometry column from the X and Y values, representing each row as a point. Then, it processes each zone separately by assigning the correct UTM CRS (EPSG:32639 or EPSG:32640), transforms the coordinates into WGS84, and stores the results. After that, the transformed GeoDataFrames are concatenated into a single dataset. Finally, the code extracts the reprojected coordinates into new columns called “Longitude” and “Latitude,” allowing the dataset to be used directly in mapping applications or spatial analyses based on global geographic coordinates.

	PLOTID	Zone	X	Y	Slope_field	Altitude_field	canopy_cover_recoded	Volume	Biomass	max_h	geometry	Longitude	Latitude
5847	A01002	39	306007	4255821	25.0	173.0	1.0	141.551240	172.277374	21.435173	POINT (306007 4255821)	48.777545	38.429600

```python
import folium
from folium.plugins import MarkerCluster

# Initialize a folium map centered around the mean location of the points
map_center = [
    data_for_SHNEI_HNEI6['Latitude'].mean()
```



GEE config

```
[ ]  from google.colab import auth
     auth.authenticate_user(project_id='prefab-hull-448203-f7')
```

```
[ ]  import ee
     ee.Authenticate()  # Run this to authenticate your account
     ee.Initialize()
```

Geometry for field data

```
[ ]  # Step 2: Create Geometry and Process Zones
     gdfs = []
     for zone in [39, 40]:  # Handle UTM zones 39 and 40
         subset = data_for_SHNFI_HNFI6[data_for_SHNFI_HNFI6['Zone'] == zone].copy()
         subset['geometry'] = subset.apply(lambda row: Point(row['X'], row['Y']), axis=1)
         crs = f"EPSG:326{zone}"  # UTM CRS for the given zone
         gdf = gpd.GeoDataFrame(subset, geometry='geometry', crs=crs)
         gdf = gdf.to_crs("EPSG:4326")  # Reproject to WGS84
         gdfs.append(gdf)

     # Combine GeoDataFrames
     combined_gdf = gpd.GeoDataFrame(pd.concat(gdfs, ignore_index=True), crs="EPSG:4326")

     # Add Longitude and Latitude columns
     combined_gdf['Longitude'] = combined_gdf.geometry.x
     combined_gdf['Latitude'] = combined_gdf.geometry.y

     # Step 3: Convert to FeatureCollection for GEE
```

In the "GEE config" section, the code sets up authentication for accessing Google Earth Engine (GEE) within a Google Colab environment.

preparing field data that includes spatial coordinates. The process involves:

- Creating geometry from X and Y coordinates.
- Assigning the correct UTM coordinate system.
- Reprojecting to WGS84 for compatibility with Earth Engine.

Meta Global Canopy Height

```
[ ]  import ee

     # Initialize Earth Engine
     ee.Initialize()

     # Load the Canopy Height ImageCollection
     canopy_collection = ee.ImageCollection('projects/meta-forest-monitoring-okw37/assets/CanopyHeight')


     # Get the first image (assuming it has the correct projection)
     first_image = canopy_collection.first()

     # Extract the projection and scale
     original_projection = first_image.projection()
     original_crs = original_projection.crs()
     original_scale = original_projection.nominalScale()

     # Mosaic and explicitly set projection to match the original image
     canopy_height = canopy_collection.mosaic().setDefaultProjection(original_crs, None, original_scale)

     # Check the resolution and projection again
     projection = canopy_height.projection()
     scale = projection.nominalScale()

     # Print final results
     print("Final Corrected Resolution (Scale) of canopy_height:", scale.getInfo(), "meters")
     print("Final Projection info:", projection.getInfo())
```

```
Final Corrected Resolution (Scale) of canopy_height: 1.194328566955889 meters
Final Projection info: {'type': 'Projection', 'crs': 'EPSG:3857', 'transform': [1.194328566955889, 0, 0, 0, -1.194328566955889, 0]}
```

Load ImageCollection

This script ensures that the **mosaicked canopy height image** retains the correct spatial resolution and projection, which is essential for accurate geospatial analysis or overlaying with other datasets. It's a common and good practice in GEE workflows when working with custom ImageCollections that need to be standardized.

ETH Global canopy height

Table of contents

Correcting the location of plots usein Meta CHM

```
[ ]  # Define grid offsets (9x9 grid with spacing of 30m, extending to ±120m)
     grid_offsets = [-120, -90, -60, -30, 0, 30, 60, 90, 120]

     # Create an empty list to store new rows
     grid_expanded_data = []

     # Iterate over each row in the original dataset
     for _, row in data_for_SHNFI_HNFI6.iterrows():
         # Loop over X and Y offsets to create a 9x9 grid
         for dx in grid_offsets:
             for dy in grid_offsets:
                 new_row = row.copy()
                 new_row['X'] = row['X'] + dx
                 new_row['Y'] = row['Y'] + dy
                 new_row['PLOTID'] = f"{row['PLOTID']}_X{dx}_Y{dy}"
                 grid_expanded_data.append(new_row)

     # Convert expanded data into a DataFrame
     grid_expanded_df = pd.DataFrame(grid_expanded_data)
```

```
[ ]  grid_expanded_df=grid_expanded_df.drop(columns=['geometry','Longitude','Latitude','Slope_field','Altitude_field','canopy_cover_recoded'])
     grid_expanded_df
```

	PLOTID	Zone	X	Y	Volume	Biomass	max_h
5847	A01002_X-120_Y-120	39	305887	4255701	141.551240	172.277374	21.435173
5847	A01002_X-120_Y-90	39	305887	4255731	141.551240	172.277374	21.435173
5847	A01002_X-120_Y-60	39	305887	4255761	141.551240	172.277374	21.435173

206631 rows × 7 columns

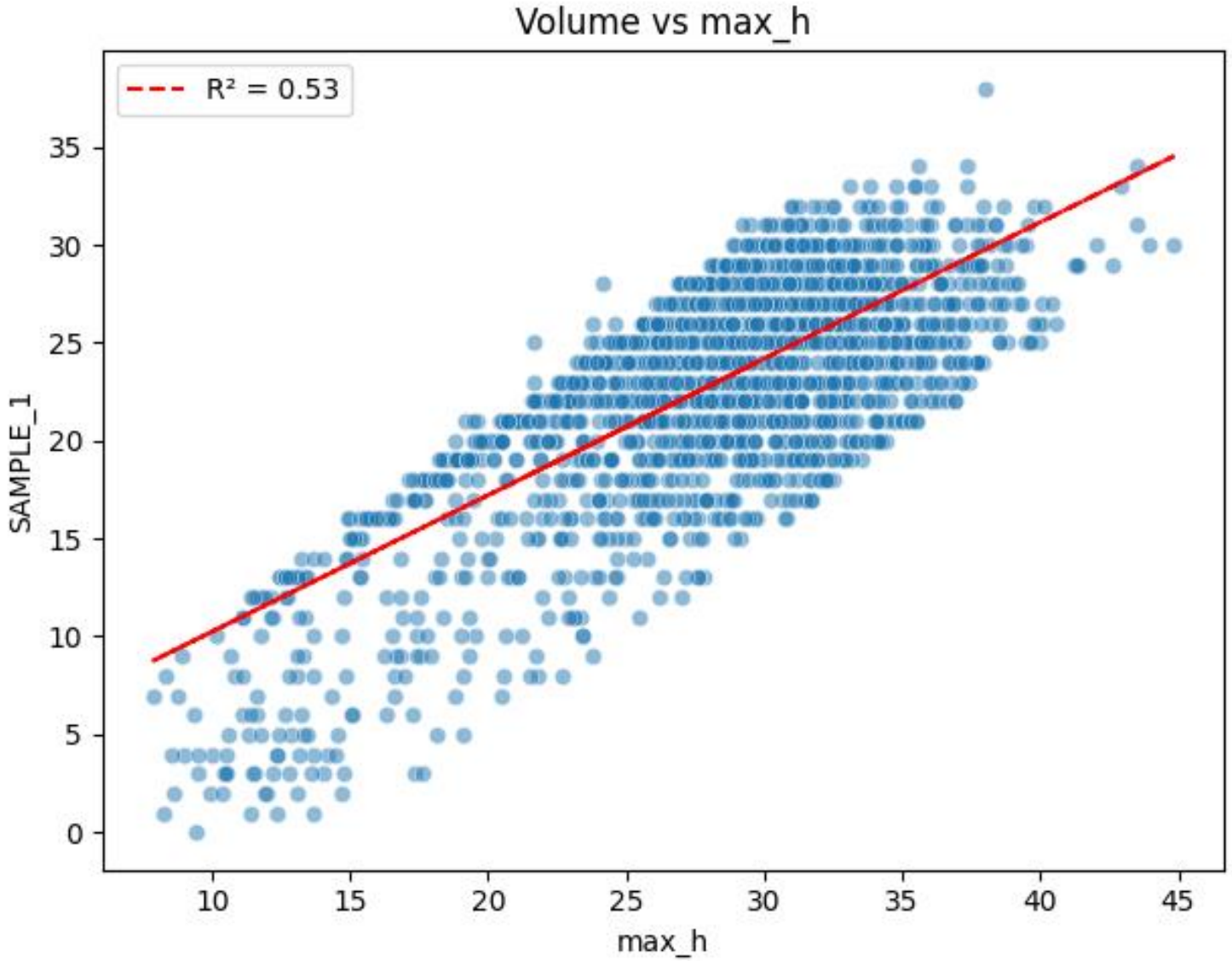
Variables Terminal

```
[ ] # Define the f
    file_path = "/

    # Read the sav
    combined_gdf_g
```

```
[ ] plt_data_location_

    plt_data_location_
```

```
[ ] plt_data_location_
```

```
Using available columns: ['max_h']
```



62	A01002_X6
128	A01003_X
204	A01004_
247	A02005_X
374	A02008_X
...	
175550	E10022_X
175663	E10023_X
175759	E10024_X
175841	E10026_X9
175856	E11002_X-1

2165 rows × 13 columr

_org	diff_h
1002	0.435173
1003	7.564227
1004	10.050113
2005	13.250099
2008	2.238410
...	...
0022	0.494454
0023	6.003358
0024	6.260630
0026	6.998118
1002	6.683028

GEDI data

Collect the data

```
# Load and filter GEDI data with time constraints
gedi = (
    ee.ImageCollection('LARSE/GEDI/GEDI02_A_002_MONTHLY')
    .select(['rh25','rh95', 'quality_flag', 'sensitivity'])  # Select RH95 and Qual
    .filterBounds(edaratkol_fc)  # Filter to study area
    .filterDate('2021-04-01', '2023-09-30')
)

# Mosaicked and clipped image
gedi_mosaic = gedi.mosaic().clip(edaratkol_fc)

# (Optional) Check how many images are in the collection
print('Number of images:', gedi.size().getInfo())

# Print metadata
print(gedi_mosaic.getInfo())
```

```
Number of images: 72
{'type': 'Image', 'bands': [{'id': 'rh25', 'data_type': {'type': 'PixelType', 'prec
```

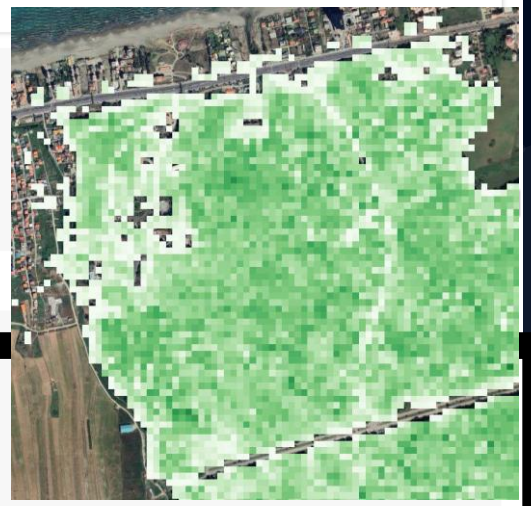
```
# Count the number of pixels (points) before filtering
point_count = gedi_mosaic.reduceRegion(
    reducer=ee.Reducer.count(),  # Count the number of valid pixels
    geometry=edaratkol_fc,  # Use the defined region
    scale=25,  # Set scale to match GEDI resolution (~25m)
    maxPixels=1e9  # Ensure large number of pixels can be counted
)

# Print the result
print("Number of points:", point_count.getInfo())
```

```
Number of points: {'quality_flag': 3532674, 'rh25': 3532674, 'rh95': 3532674, 'sens
```

These data need to be filtered into high-quality data.

Filter by quality and outliers

```
[ ]  # Create a mask to keep only high-quality data (quality_flag == 1)
     quality_mask = gedi_mosaic.select('quality_flag').eq(1).And(
         gedi_mosaic.select('sensitivity').gt(0.95)
     )

     # Apply the mask
     filt_qlt_gedi = gedi_mosaic.updateMask(quality_mask)
```



```
[ ]  # Count the number of pixels (points) in the valid forest height region
     point_count = valid_forest_height.reduceRegion(
         reducer=ee.Reducer.count(),  # Count the number of valid pixels
         geometry=edaratkol_fc,  # Use the defined region
         scale=25,  # Set scale to match GEDI resolution (~25m)
         maxPixels=1e9  # Ensure large number of pixels can be counted
     )

     # Print the result
     print("Number of valid forest points:", point_count.getInfo())
```

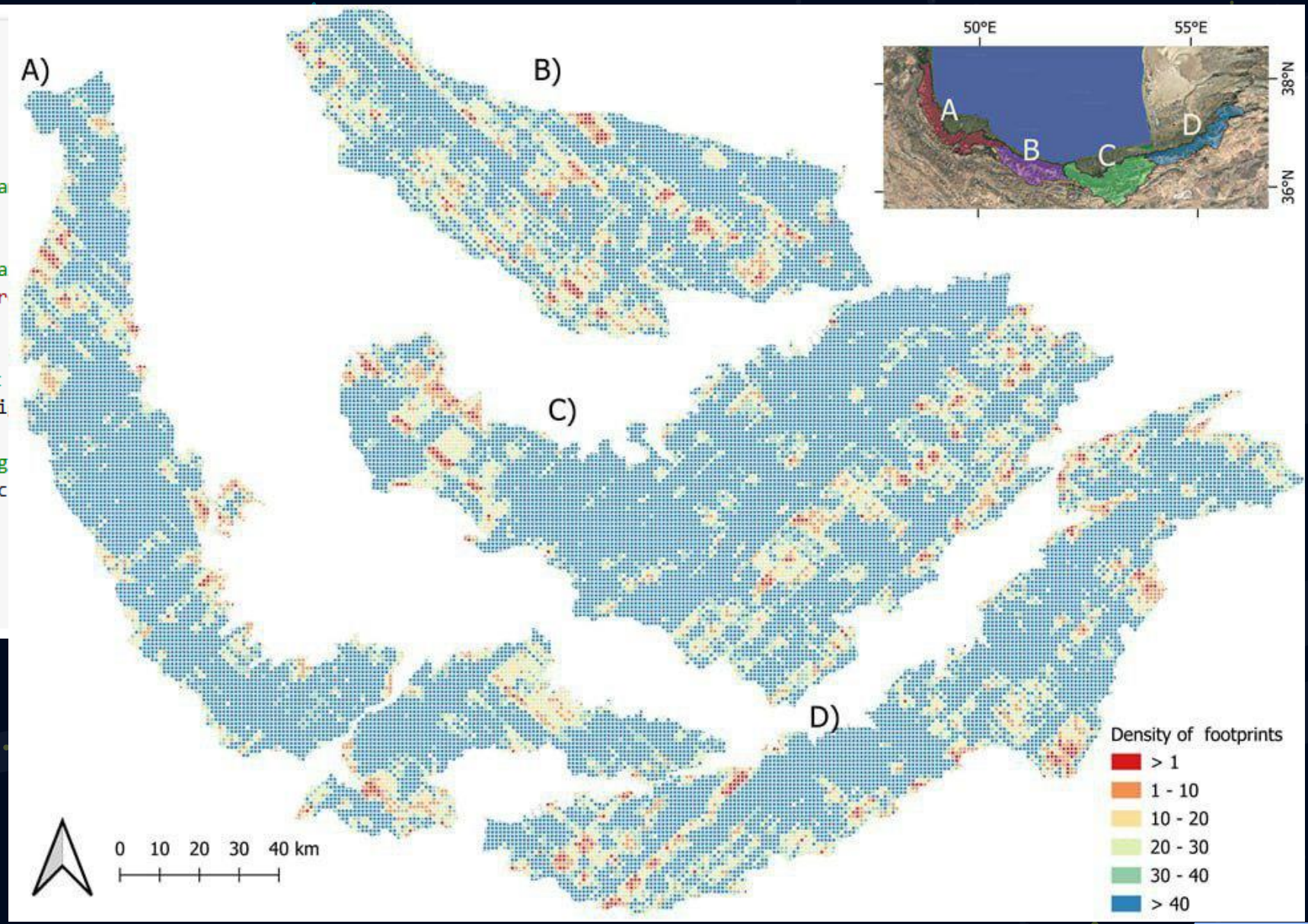
```
Number of valid forest points: {'quality_flag': 601744, 'rh95': 730624, 'sensitivity': 992219}
```

```
[ ]  # Compute the difference and rename the band
     diff_image = valid_forest_height.select('rh95') \
                     .subtract(valid_forest_height.select('rh25')) \
                     .rename('diff_rh95_rh25')

     # Add the new band to the existing image
     valid_forest_height_with_diff = valid_forest_height.addBands(diff_image)

     # (Optional) Print info about the updated image
     print(valid_forest_height_with_diff.getInfo())
```

```
{'type': 'Image', 'bands': [{'id': 'rh25', 'data_type': {'type': 'PixelType', 'precision': 'double'}, 'crs': 'EPSG:4326', 'crs_transform': [1, 0, 0, 0, 1, 0]}, {'id': 'rh95', 'data_type': {'type': 'Pixel
```

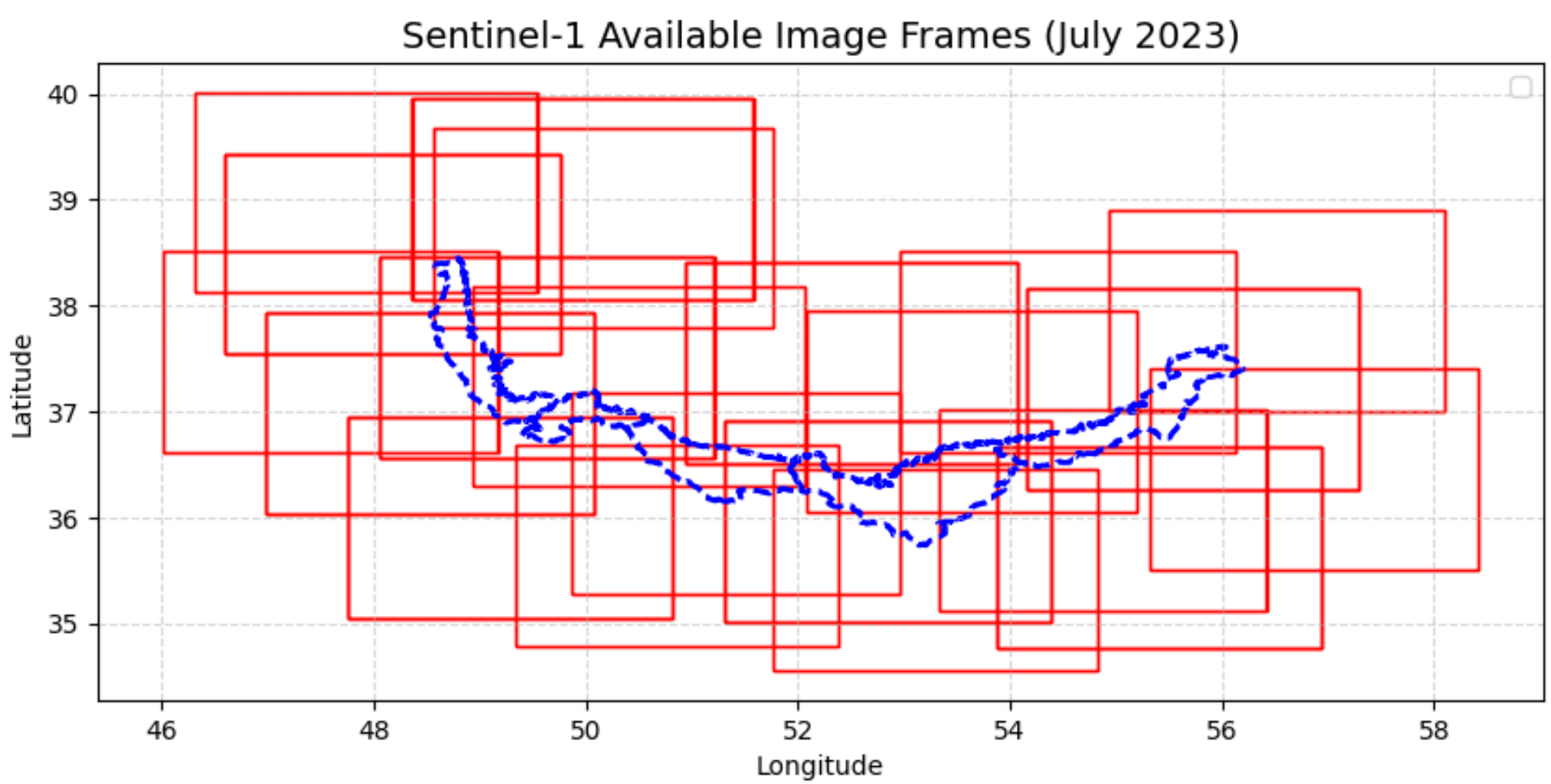
And then add to map with this code

```
from geemap import Map

# Initialize the map
m = Map()

# Define visualization para
gedi_vis_params = {
    'min': 0,
    'max': 50,  # Adjust ma
    'palette': ['blue', 'gr
}

# ✅ Correct way to select
m.addLayer(valid_forest_hei

# Center the map on the reg
m.centerObject(edaratkol_fc

# Display the map
m
```



Sentinel 1

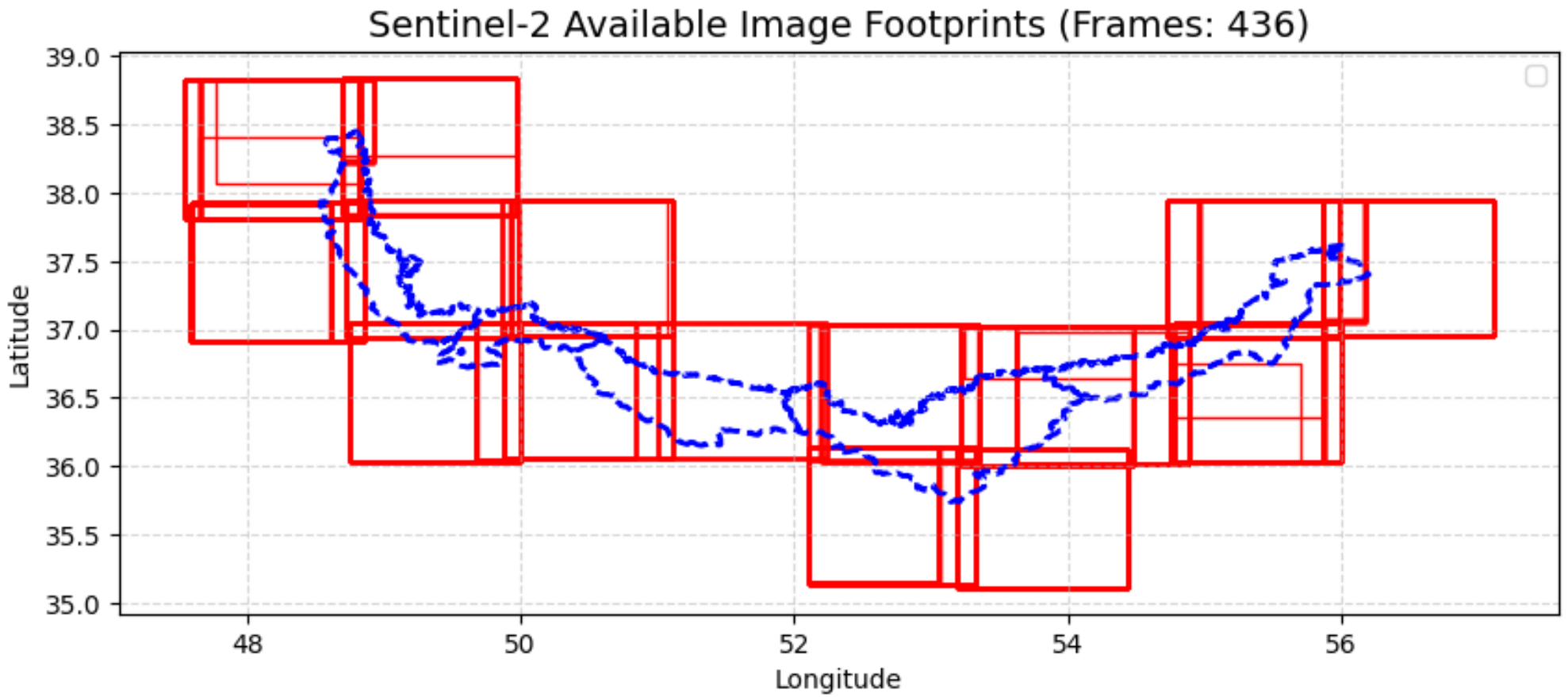
```
# Show the static plot
plt.show()
```



Sentinel 2

```
# Show the static plot
plt.show()
```

```
✅ Number of Sentinel-2 Frames: 436
```



$$NDVI = \frac{(NIR - RED)}{(NIR + RED)}$$

- Canopy Chlorophyll Content Index (CCCI) Approximation

$$CCCI = \frac{(NIR - SWIR1)}{(NIR + SWIR1)}$$

  - Uses SWIR1 instead of the true Red Edge band (Landsat lacks a dedicated Red Edge band).
- Enhanced Vegetation Index (EVI)

$$EVI = 2.5 \times \frac{(NIR - RED)}{(NIR + 6 \times RED - 7.5 \times BLUE + 1)}$$

- Soil-Adjusted Vegetation Index (SAVI)

$$SAVI = \left(\frac{(NIR - RED)}{(NIR + RED + 0.5)}\right) \times (1 + 0.5)$$

- Modified Chlorophyll Absorption in Reflectance Index (MCARI2) Approximation

$$MCARI2 = ((SWIR1 - RED) - 0.2 \times (SWIR1 - BLUE)) \times \frac{SWIR1}{RED}$$

```
[ ] # Define bands for calculations using renamed Sentinel-2 bands
    band_map = {
        'NIR': s2_mosaic.select('S2_B8'),   # Near Infrared
        'RED': s2_mosaic.select('S2_B4'),   # Red
        'GREEN': s2_mosaic.select('S2_B3'), # Green
        'BLUE': s2_mosaic.select('S2_B2'),  # Blue
        'RE': s2_mosaic.select('S2_B5')     # Red Edge
    }

    # Compute vegetation indices
    indices_s2 = ee.Image([
        s2_mosaic.normalizedDifference(['S2_B8', 'S2_B4']).rename('S2_NDVI'),  # NDVI: (NIR - Red) / (NIR + Red)
        s2_mosaic.expression('(NIR - RE) / (NIR + RE)', band_map).rename('S2_CCCI'), # CCCI: (NIR - RE) / (NIR + RE)
        s2_mosaic.expression('2.5 * ((NIR - RED) / (NIR + 6 * RED - 7.5 * BLUE + 1))', band_map).rename('S2_EVI'), # EVI
        s2_mosaic.expression('((NIR - RED) / (NIR + RED + 0.5)) * (1 + 0.5)', band_map).rename('S2_SAVI'), # SAVI
        s2_mosaic.expression('((RE - RED) - 0.2 * (RE - BLUE)) * (RE / RED)', band_map).rename('S2_MCARI2') # MCARI2
    ])
```

Vegetation indices are spectral metrics derived from satellite or airborne remote sensing data that provide a quantitative measure of vegetation condition and dynamics. They are typically calculated by combining reflectance values from specific wavelength bands, most commonly the red and near-infrared regions, to capture the distinct spectral behavior of healthy vegetation, which absorbs strongly in the red for photosynthesis and reflects highly in the near-infrared due to leaf structure. Through these mathematical transformations, indices such as the Normalized Difference Vegetation Index (NDVI) enable researchers to assess vegetation vigor, density, and stress levels, as well as to monitor temporal changes in biomass and ecosystem functioning. Their simplicity, robustness, and strong ecological relevance have made vegetation indices fundamental tools in forest monitoring, agriculture, and environmental studies worldwide.

Visvalize the mosaic

[ ] ↳ 1 cell hidden

Compile S2 and GEDI

```
[ ] import ee
    import pandas as pd
```

	Longitude	Latitude	RH95	S2_B2	S2_B3	S2_B4	S2_B5	\
0	49.109055	37.442634	2.09	0.063729	0.091991	0.110318	0.137272	
1	53.763766	36.276531	12.96	0.073792	0.092871	0.106673	0.139820	
2	53.600991	36.416938	5.66	0.048923	0.061522	0.096523	0.132567	
3	49.005030	37.394934	18.90	0.006229	0.050191	0.020818	0.107172	
4	53.215075	36.213739	14.82	0.004380	0.023426	0.007328	0.077277	

	S2_B6	S2_B7	S2_B8	S2_B8A	S2_B11	S2_B12	S2_NDVI	\
0	0.159041	0.172652	0.165084	0.167557	0.069285	0.039487	0.198857	
1	0.279844	0.312382	0.330373	0.351353	0.326605	0.221934	0.511846	
2	0.233988	0.267710	0.272544	0.287481	0.267124	0.189676	0.476937	
3	0.363741	0.452652	0.474884	0.477357	0.215885	0.098887	0.916006	
4	0.360913	0.449904	0.464224	0.481139	0.237847	0.099949	0.968921	

	S2_CCCI	S2_EVI	S2_SAVI	S2_MCARI2
0	0.091985	0.101491	0.105943	0.015237
1	0.405266	0.394681	0.358094	0.026138
2	0.345527	0.296381	0.303812	0.026529
3	0.631748	0.730915	0.684039	0.340618
4	0.714581	0.774223	0.705412	0.583926

```
# Convert to a FeatureCollection and extract data
features = sampled_s2.getInfo().get('features', [])  # Use .get() to avoid KeyError

# Convert extracted data to a Pandas DataFrame
```

```
g keys exist in properties

', None),
B2', None),   # Blue
B3', None),   # Green
B4', None),   # Red
B5', None),   # Red Edge 1
B6', None),   # Red Edge 2
B7', None),   # Red Edge 3
B8', None),   # NIR
_B8A', None), # Narrow NIR
_B11', None), # SWIR1
_B12', None), # SWIR2
2_NDVI', None),
2_CCCI', None),
_EVI', None),
2_SAVI', None),
'S2_MCARI2', None)

DataFrame(data)
```

Variables Terminal

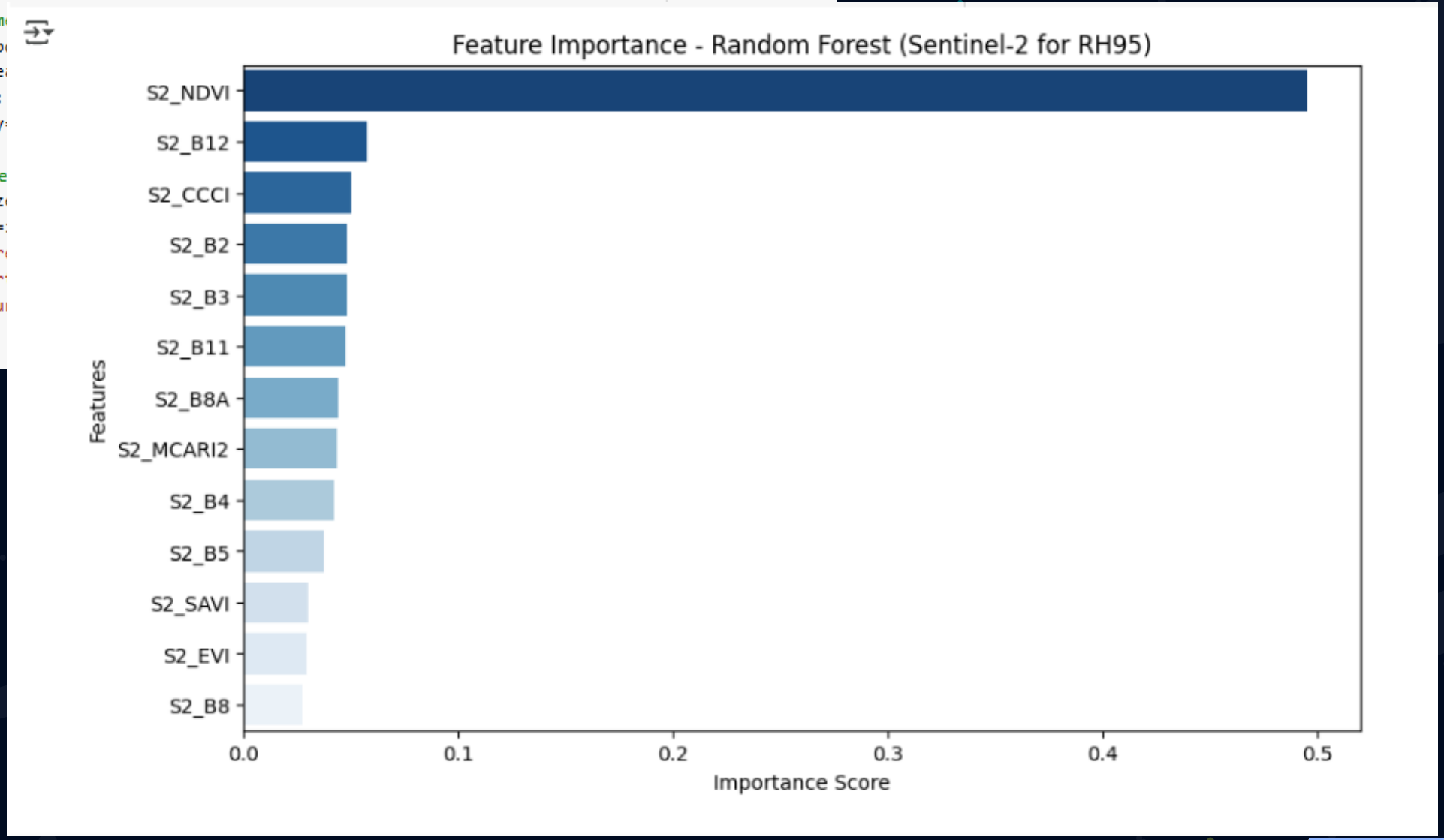
In the next step, generating maps and conducting the required analyses involves combining certain images and datasets. Since this process is highly specialized and time-consuming, we will only provide a brief mention of it here and move forward without delving into the technical details.

Random Forest Algorithm


```
[ ]  # Create a DataFrame
     ml_training_df = pd.DataFrame(gedi_sentinel_Spectral_df)

     # Step 5: Train the Random Forest Model Again
     from sklearn.model_selection import train_test_split
     from sklearn.ensemble import RandomForestRegressor
     from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
     import numpy as np

     # Drop any rows with missing values
     df = ml_training_df.dropna()

     # Define features and target variable
     features = ['S2_B2', 'S2_B3', 'S2_B4', 'S2_B5', 'S2_B8', 'S2_B8A', 'S2_B11', 'S2_B12',
                 'S2_NDVI', 'S2_CCCI', 'S2_EVI', 'S2_SAVI', 'S2_MCARI2']
     target = 'RH95'

     # Split the dataset (70% training, 30% testing)
     X_train, X_test, y_train, y_test = train_test_split(df[features], df[target], test_size=0.3, random_state=42)

     # Train the Random Forest model
     rf_model_sentinel2 = RandomForestRegressor(n_estimators=100, random_state=42)
     rf_model_sentinel2.fit(X_train, y_train)

     # Predict on test data
     y_pred = rf_model_sentinel2.predict(X_test)

     # Evaluate Model Performance
     r2 = r2_score(y_test, y_pred)
     rmse = np.sqrt(mean_squared_error(y_test, y_pred))
     mae = mean_absolute_error(y_test, y_pred)

     # Step 6: Display Model Performance Results
     results_S2_RH95_df = pd.DataFrame({
         "Metric": ["R² Score", "RMSE", "MAE"],
         "Value": [r2, rmse, mae]
     })
     results_S2_RH95_df
```

	Metric	Value
0	R² Score	0.391617
1	RMSE	9.152890
2	MAE	6.705117

```
[ ]  import matplotlib.pyplot as plt
     import seaborn as sns
     import pandas as pd

     # Get feature importances
     feature_importances = rf_model_sentinel2.feature_importances_

     # Create DataFram
     importance_df = p
         "Feature": fe
         "Importance":
     }).sort_values(by

     # 📊 Plot feature
     plt.figure(figsiz
     sns.barplot(data=
     plt.title("Featur
     plt.xlabel("Impor
     plt.ylabel("Featu
     plt.show()
```



```python
# Evaluate Model Performance
r2 = r2_score(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = mean_absolute_error(y_test, y_pred)

# Calculate RMSE%
rmse_percent = (rmse / (y_test.max() - y_test.min())) * 100

# Calculate RMSE based on the mean
mean_target = y_test.mean()
rmse_based_on_mean = np.sqrt(np.mean((y_pred - mean_target) ** 2))


# Evaluate Model Performance
r2 = r2_score(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = mean_absolute_error(y_test, y_pred)



# Step 6: Display Model Performance Results
results_Textural_RH95_df = pd.DataFrame({
    "Metric": ["R² Score", "RMSE", "RMSE% (based on mean)", "RMSE% (based on range)","MAE"],
    "Value": [r2, rmse, rmse_based_on_mean, rmse_percent, mae]
})

results_Textural_RH95_df
```

	Metric	Value
0	R² Score	0.480514
1	RMSE	8.283621
2	RMSE% (based on mean)	7.969259
3	RMSE% (based on range)	14.489453
4	MAE	6.224258

$$\text{RMSE (based on mean)} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_{\text{pred}}^{i} - \bar{y})^2}$$

Where:

- y_{pred}^{i} is the predicted value for the i^{th} data point,
- $\bar{y}$ is the mean of the actual target values (`y_test`),
- n is the number of samples.

1. **RMSE% Calculation:**
   - We calculate the **range** of `RH95` in the test set (`y_test.max() - y_test.min()`).
   - Then, the RMSE percentage is calculated as:

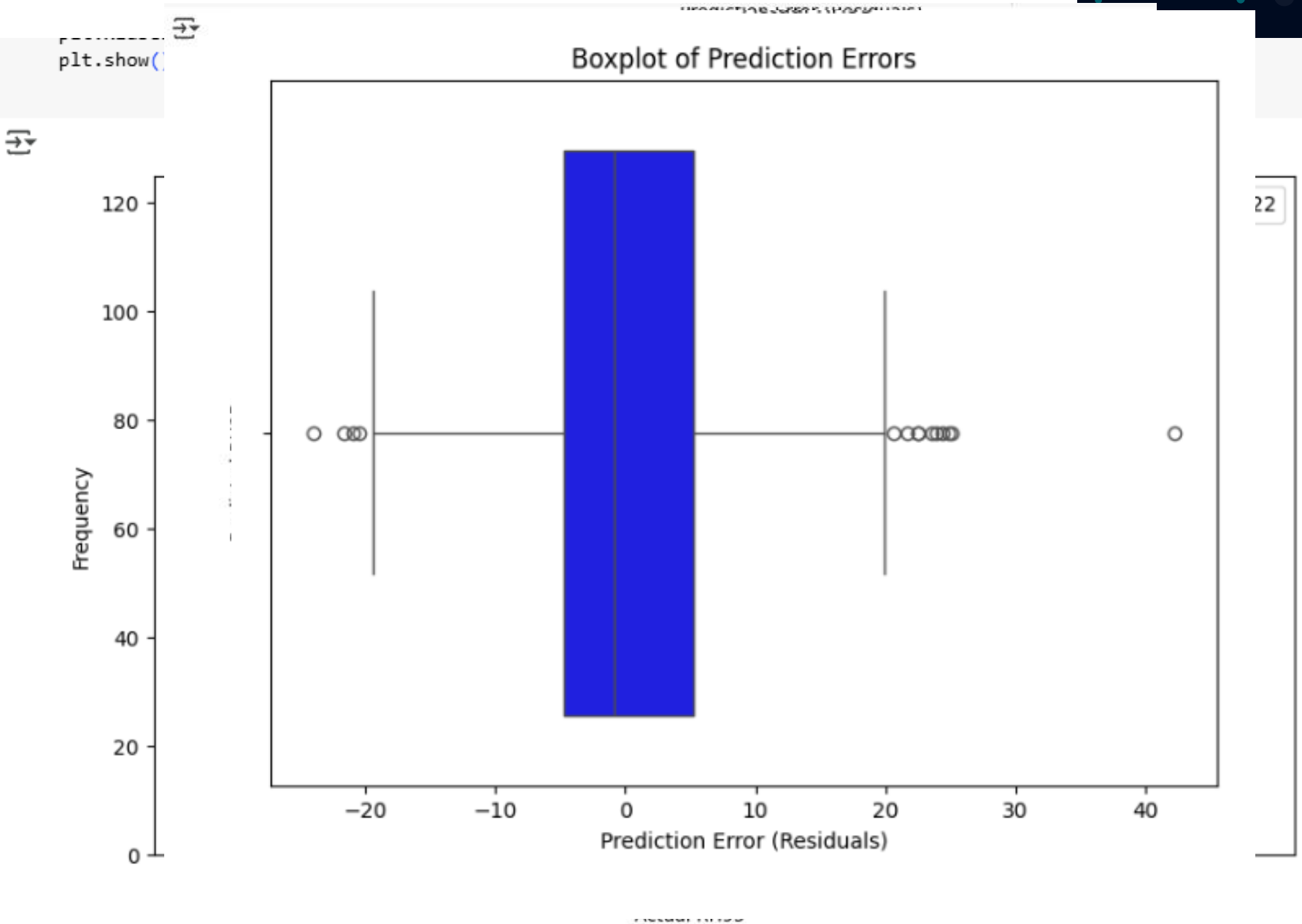
$$\text{RMSE\%} = \frac{\text{RMSE}}{\text{Range of RH95}} \times 100$$

This code snippet evaluates the performance of a predictive model using multiple statistical metrics. First, it calculates the coefficient of determination (R^2), root mean square error (RMSE), and mean absolute error (MAE) to assess how well the predicted values match the observed ones. It also computes RMSE expressed as a percentage relative to both the data range and the mean target value, providing normalized measures of error that facilitate comparison across different datasets. Finally, the results are organized into a dataframe for clear presentation, showing values for R^2, RMSE, RMSE% (based on mean and range), and MAE, which together provide a comprehensive overview of the model's accuracy and reliability.

Diagnosis

```
[ ]  import matplotl
     import seaborn

     # Create a resi
     errors = y_test

     # Plot the erro
     plt.figure(figs
     sns.histplot(er
     plt.axvline(err
     plt.xlabel("Pre
     plt.ylabel("Fre
     plt.title("Erro
     plt.legend()
     plt.show()

     # Plot actual v
     plt.figure(figs
     sns.scatterplot
     plt.plot([min(y
     plt.xlabel("Act
     plt.ylabel("Pre
     plt.title("Actu
     plt.show()

     # Boxplot to vi
     plt.figure(figs
     sns.boxplot(x=e
     plt.title("Boxp
     plt.xlabel("Pre
     plt.show()
```

```
plt.show()
```



The Hyrcanian forests are characterized by steep and complex slopes, and to address this challenge and achieve more accurate estimations, we utilize SRTM data.

SRTM

Collect the data

```
[ ] import ee

    # Initialize Earth Engine
    ee.Initialize()

    # Load SRTM DEM
    srtm = ee.Image("USGS/SRTMGL1_003")

    # Clip the DEM to the region of interest (ROI)
    srtm_clipped = srtm.clip(edaratkol_fc)

    # Convert to Float and Mask No-Data (-32768)
    srtm = srtm_clipped.toFloat().updateMask(srtm_clipped.neq(-32768))

    # Reduce Region with Proper Parameters
    stats = srtm.reduceRegion(
        reducer=ee.Reducer.minMax(),
        geometry=edaratkol_fc,
        scale=500,  # Increase scale to reduce pixel count
        bestEffort=True,  # Allows reducing the number of pixels automatically
        maxPixels=1e8  # Increase pixel limit to avoid errors
    )

    # Print Fixed Min/Max Values
    print("SRTM Min/Max after fixing:", stats.getInfo())
```

```
SRTM Min/Max after fixing: {'elevation_max': 4673, 'elevation_min': -26}
```

Calculation

```
[ ] import ee

    # Initialize Earth Engine
    ee.Initialize()

    # **1. Load & Clip SRTM DEM**
    srtm = ee.Image("USGS/SRTMGL1_003").clip(edaratkol_fc)

    # **2. Convert DEM to Float & Fill Sinks**
    srtm_float = srtm.toFloat()

    # **3. Compute Slope**
    slope = ee.Terrain.slope(srtm).clip(edaratkol_fc)

    # **4. Compute Aspect**
    aspect = ee.Terrain.aspect(srtm).clip(edaratkol_fc)


    # **10. Print Metadata**
    print("Slope:", slope.getInfo())
    print("Aspect:", aspect.getInfo())
```

```
Slope: {'type': 'Image', 'bands': [{'id': 'slope', 'data_type': {'type': 'PixelType', 'precision': 'float', 'min': 0, 'max': 90}, 'crs': 'EPSG:4326', 'crs_transform': [0.0002777777777777778, 0, -180.000138888888
Aspect: {'type': 'Image', 'bands': [{'id': 'aspect', 'data_type': {'type': 'PixelType', 'precision': 'float', 'min': 0, 'max': 360}, 'crs': 'EPSG:4326', 'crs_transform': [0.0002777777777777778, 0, -180.000138888
```

```
[ ]  # Ensure PLOTID is the same datatype in both DataFrames
     df_sampled["PLOTID"] = df_sampled["PLOTID"].astype(str)
     data_for_SHNFI_HNFI6["PLOTID"] = data_for_SHNFI_HNFI6["PLOTID"].astype(str)

     # Merge both DataFrames on PLOTID
     final_data = pd.merge(data_for_SHNFI_HNFI6, df_sampled, on="PLOTID", how="left")

     # Display first rows of merged DataFrame
     print(final_data.head())
```

```
    PLOTID  Zone       X        Y  Slope_field  Altitude_field  \
0  A01002    39  306007  4255821         25.0           173.0
1  A01003    39  307007  4255821         35.0            51.0
2  A01004    39  308007  4255821         40.0            73.0
3  A02005    39  293007  4250821         30.0           673.0
4  A02008    39  296007  4250821         75.0           718.0

   canopy_cover_recoded      Volume     Biomass      max_h  ...    S2_B8A  \
0                   1.0  141.551240  172.277374  21.435173  ...  0.245675
1                   1.0  186.147186  207.621682  23.564227  ...  0.264275
```

final_data

	PLOTID	Zone	X	Y	Slope_field	Altitude_field	canopy_cover_recoded	Volume	Bi
0	A01002	39	306007	4255821	25.0	173.0	1.0	141.551240	172.2
1	A01003	39	307007	4255821	35.0	51.0	1.0	186.147186	207.
2	A01004	39	308007	4255821	40.0	73.0	3.0	59.602802	68.
3	A02005	39	293007	4250821	30.0	673.0	3.0	268.831734	206.
4	A02008	39	296007	4250821	75.0	718.0	2.0	188.034184	199.
...	...	...	...	...	...	...	...	...	
2548	E10022	39	460008	4079821	30.0	835.0	1.0	433.896533	320.
2549	E10023	39	460008	4080821	60.0	833.0	1.0	259.255345	252.
2550	E10024	39	460008	4081821	60.0	867.0	1.0	312.422423	277.
2551	E10026	39	459993	4083880	20.0	1218.0	3.0	203.638813	164.
2552	E11002	39	455008	4075821	35.0	2113.0	1.0	629.680544	517.

2553 rows × 49 columns

In this section, all datasets are merged together, including those derived from remote sensing imagery and the field-based data obtained through forest inventory. These are integrated into a single dataframe, the column headers of which are presented here.

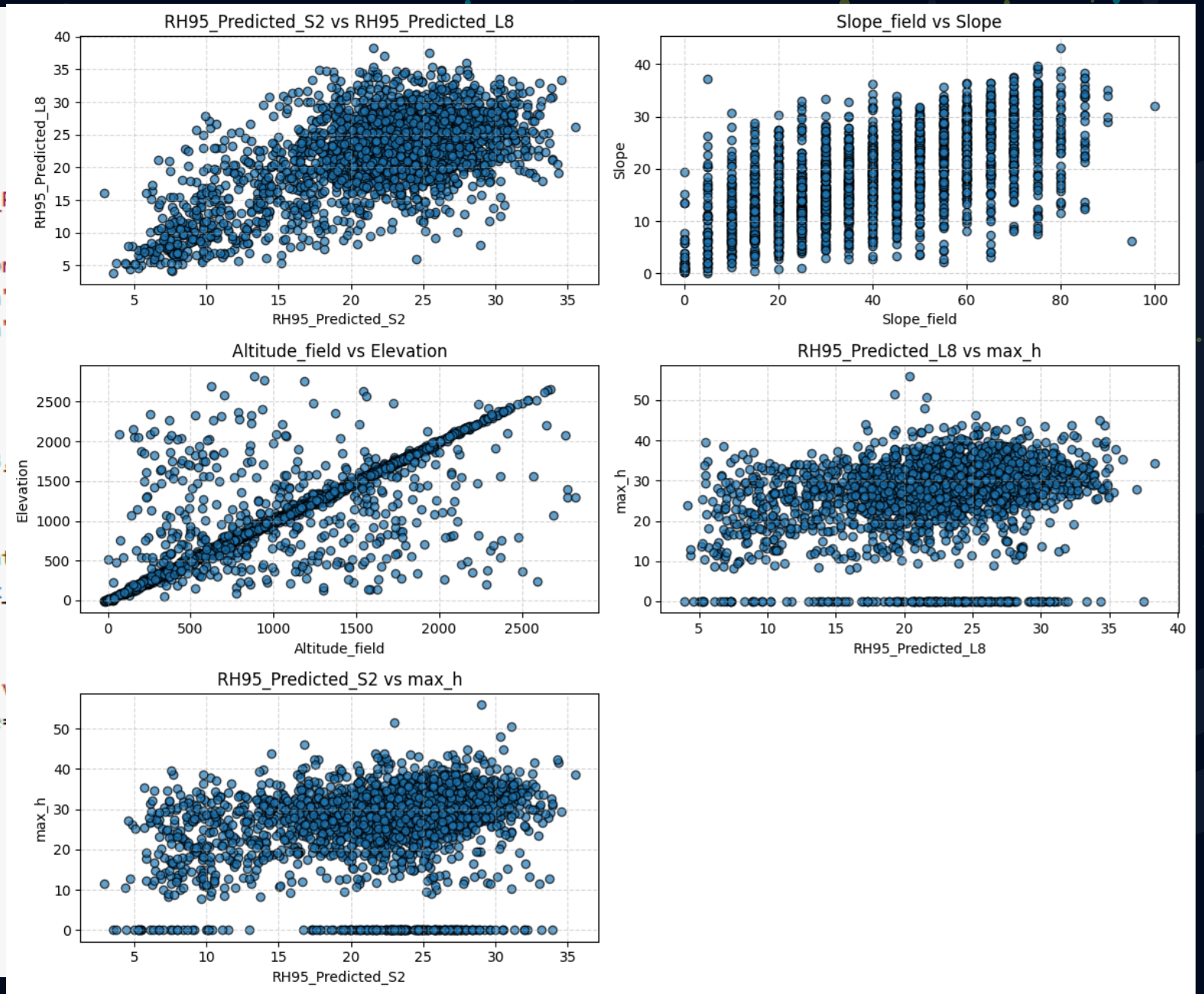
```
final_data.columns
```

```
Index(['PLOTID', 'Zone', 'X', 'Y', 'Slope_field', 'Altitude_field',
       'canopy_cover_recoded', 'Volume', 'Biomass', 'max_h', 'geometry',
       'Longitude_x', 'Latitude_x', 'Longitude_y', 'Latitude_y', 'Aspect',
       'Elevation', 'L8_B2', 'L8_B3', 'L8_B4', 'L8_B5', 'L8_B6', 'L8_B7',
       'L8_CCCI', 'L8_EVI', 'L8_MCARI2', 'L8_NDVI', 'L8_SAVI', 'S2_B11',
       'S2_B12', 'S2_B2', 'S2_B3', 'S2_B4', 'S2_B5', 'S2_B6', 'S2_B7', 'S2_B8',
       'S2_B8A', 'S2_CCCI', 'S2_EVI', 'S2_MCARI2', 'S2_NDVI', 'S2_SAVI',
       'Slope', 'TPI', 'VH', 'VV'],
      dtype='object')
```

```
[ ]  final_data.columns
```

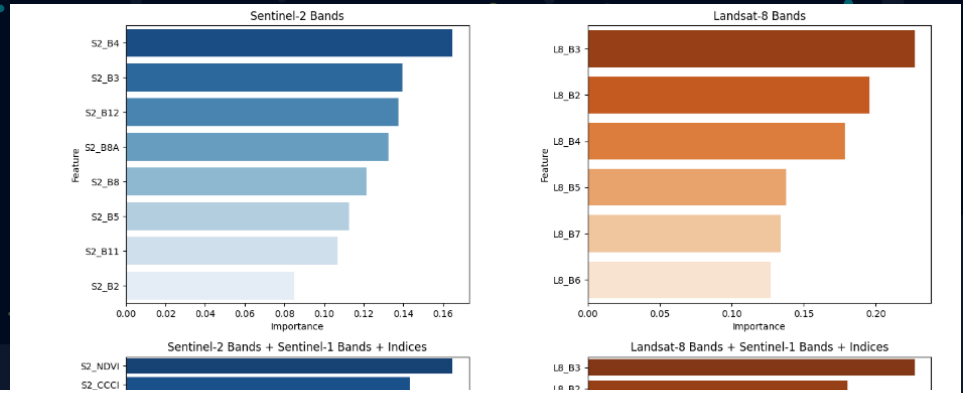
```
Index(['PLOTID', 'Zone', 'X', 'Y', 'Slope_field', 'Altitude_field',
       'canopy_cover_recoded', 'Volume', 'Biomass', 'max_h', 'geometry',
       'Longitude_x', 'Latitude_x', 'Longitude_y', 'Latitude_y', 'Aspect',
       'Elevation', 'L8_B2', 'L8_B3', 'L8_B4', 'L8_B5', 'L8_B6', 'L8_B7',
       'L8_CCCI', 'L8_EVI', 'L8_MCARI2', 'L8_NDVI', 'L8_SAVI', 'S2_B11',
       'S2_B12', 'S2_B2', 'S2_B3', 'S2_B4', 'S2_B5', 'S2_B6', 'S2_B7', 'S2_B8',
       'S2_B8A', 'S2_CCCI', 'S2_EVI', 'S2_MCARI2', 'S2_NDVI', 'S2_SAVI',
       'Slope', 'TPI', 'VH', 'VV', 'RH95_Predicted_S2', 'RH95_Predicted_L8'],
      dtype='object')
```

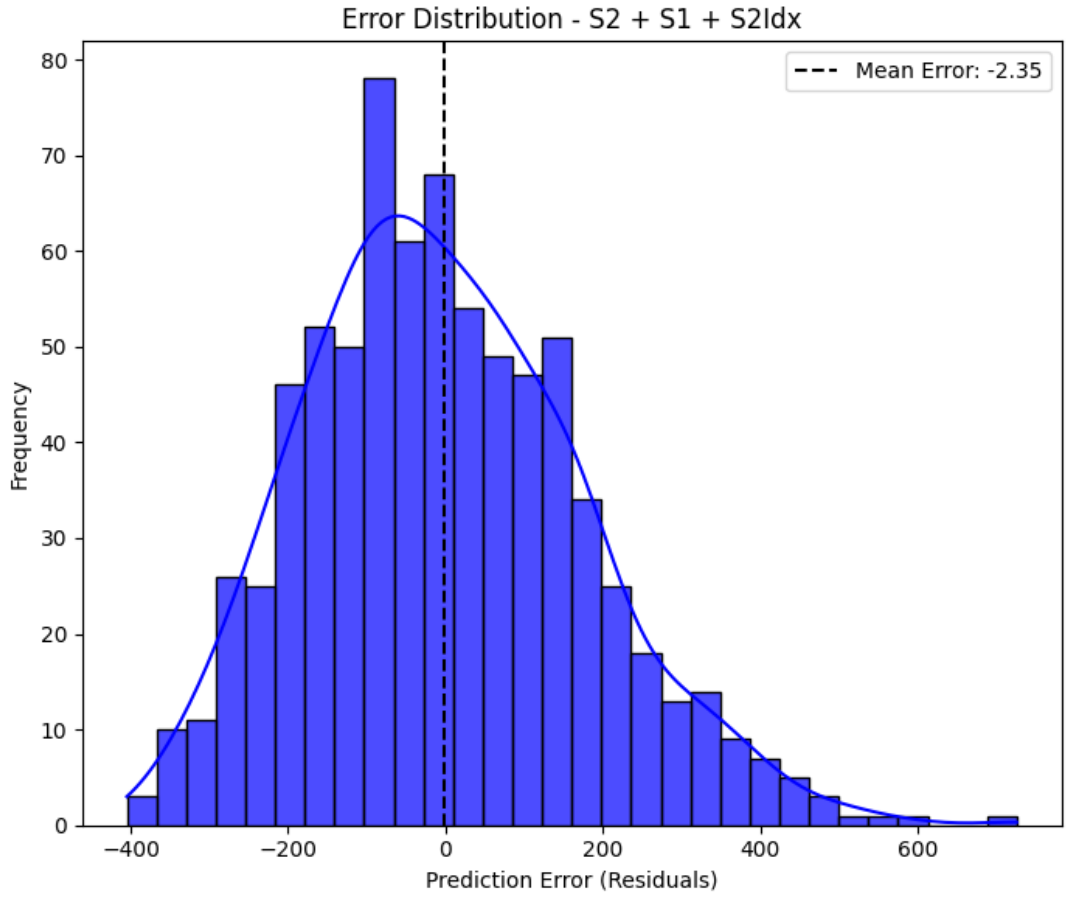
```
[ ]  import matplotlib.pyplot as plt
     import pandas as pd

     # Define scatter plots
     scatter_plots = [
         ("RH95_Predicted_S2", "RH95_
         ("Slope_field", "Slope"),
         ("Altitude_field", "Elevatio
         ("RH95_Predicted_L8", "max_h
         ("RH95_Predicted_S2", "max_h
     ]

     # Generate scatter plots
     fig, axes = plt.subplots(nrows=3,
     axes = axes.flatten()

     for i, (x_col, y_col) in enumerat
         axes[i].scatter(final_data[x_
         axes[i].set_xlabel(x_col)
         axes[i].set_ylabel(y_col)
         axes[i].set_title(f"{x_col}
         axes[i].grid(True, linestyle=

     # Remove any empty subplot
     fig.delaxes(axes[-1])

     # Adjust layout
     plt.tight_layout()
     plt.show()
```

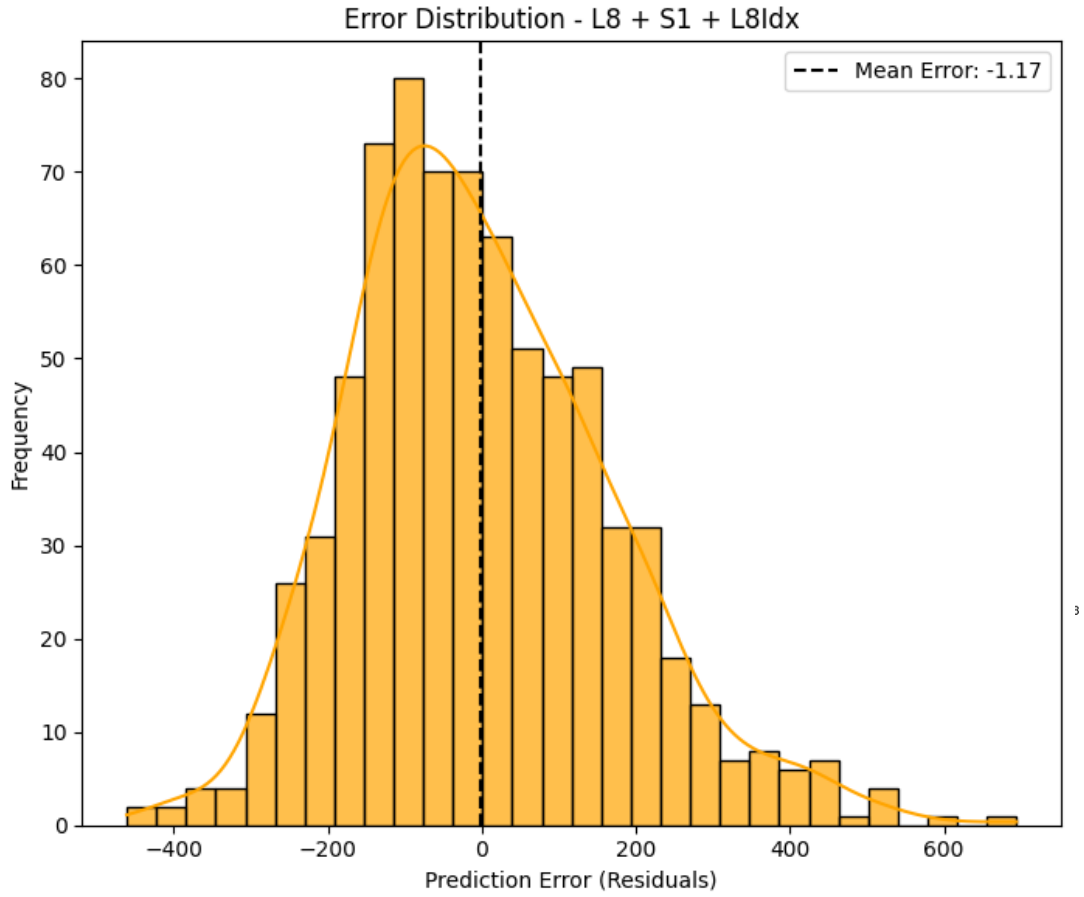


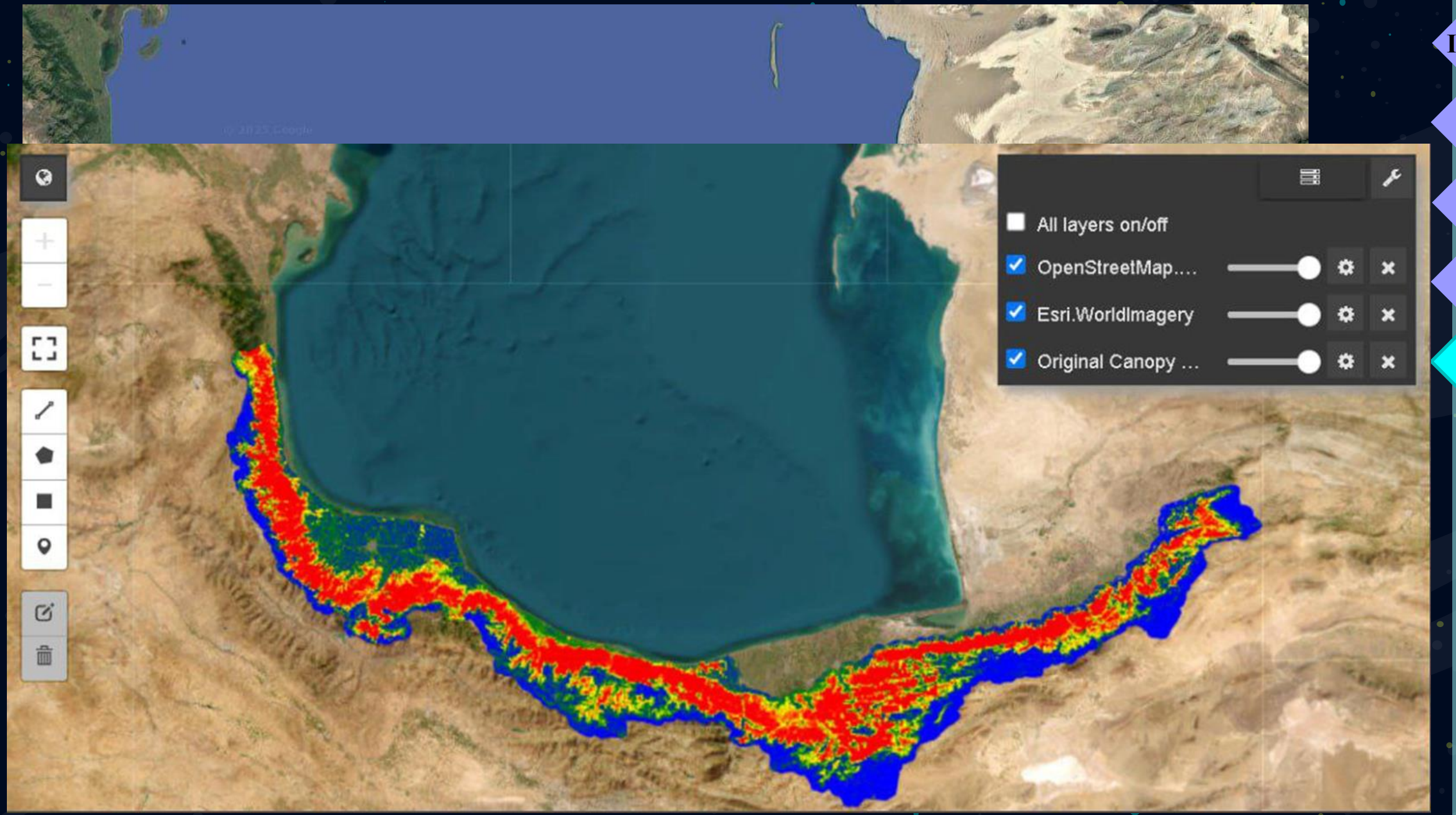
```
# Convert results to DataFrame
results_df = pd.DataFrame(results)
results_df
```

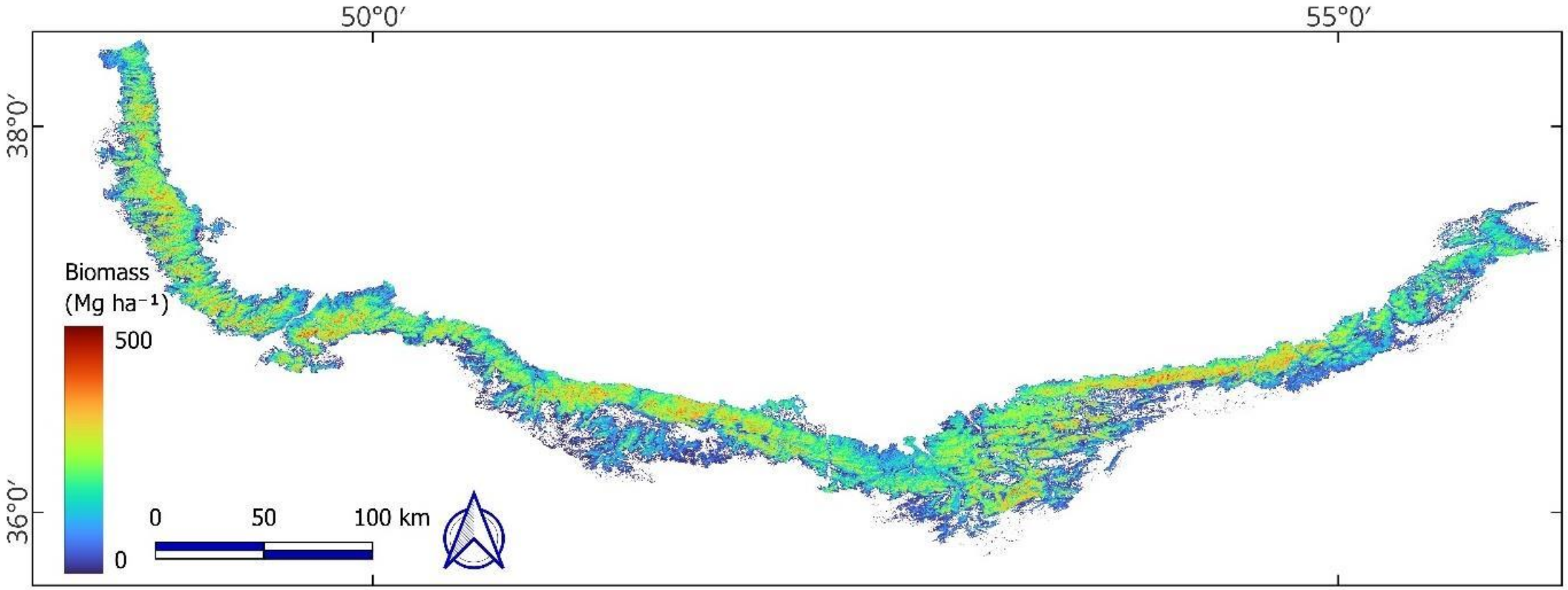
	Feature Set	R^2 Score	RMSE	MAE
0	Sentinel-2 Bands	0.085750	192.379801	151.396364
1	Sentinel-2 Bands + Sentinel-1 Bands	0.109119	189.905139	149.197437








All layers on/off
OpenStreetMap....
Esri.WorldImagery
Original Canopy ...



Laya Zeinali Yadegari [a], Hormoz Sohrabi [a], Elia Quirós [b], Markus Immitzer [c]

Q & A

Laya Zeinali
layazeinaliyadegari@gmail.com
Linkedln: Laya Zeinali

THANKS FOR YOUR ATTENTION